Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)**

Кафедра технологий электронного обучения (ТЭО)

МАТЕМАТИЧЕСКОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ВЫЧИСЛИТЕЛЬНЫХ МАШИН, КОМПЛЕКСОВ И КОМПЬЮТЕРНЫХ СЕТЕЙ

Методические указания к практическим занятиям
и организации самостоятельной работы аспирантов

2018

Кручинин Владимир Викторович, Морозова Юлия Викторовна
Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей: Методические указания к практическим занятиям и организации самостоятельной работы аспирантов / В.В. Кручинин, Ю.В. Морозова. – Томск, 2018. – 72 с.

Оглавление

1 Введение

Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей – специальность, включающая задачи развития теории программирования, создания и сопровождения программных средств различного назначения. Научное и народнохозяйственное значение решения проблем данной специальности состоит в повышении эффективности и надежности процессов обработки и передачи данных и знаний в вычислительных машинах, комплексах и компьютерных сетях.

К области исследований этого направления относятся

1. Модели, методы и алгоритмы проектирования и анализа программ и программных систем, их эквивалентных преобразований, верификации и тестирования.

2. Языки программирования и системы программирования, семантика программ.

3. Модели, методы, алгоритмы, языки и программные инструменты для организации взаимодействия программ и программных систем.

4. Системы управления базами данных и знаний.

5. Программные системы символьных вычислений.

6. Операционные системы.

7. Человеко-машинные интерфейсы; модели, методы, алгоритмы и программные средства машинной графики, визуализации, обработки изображений, систем виртуальной реальности, мультимедийного общения.

8. Модели и методы создания программ и программных систем для параллельной и распределенной обработки данных, языки и инструментальные средства параллельного программирования.

9. Модели, методы, алгоритмы и программная инфраструктура для организации глобально распределенной обработки данных.

2 Методические указания для проведения практических занятий

2.1 Практическое занятие «Системный анализ. Разработка требований к алгоритмическому и программному обеспечению»

Теоретические основы

Программирование – это процесс создания программ для ЭВМ. На заре компьютерной эры программирование воспринималось как искусство. Однако, по мере развития систем программирования этот процесс становится все более изученным. Появляются термины *методология программирования*, *технология программирования, наука программирования инженерия программного обеспечения*. Однако, в споре о том, что программирование – это искусство или наука нет окончательного решения.

Основным, определяющим элементом современных систем программирования является методология. Методология, как философская категория, это система принципов и основ организации и построения творческой и практической деятельности. Таким образом, *методология* программирования есть система принципов, основ организации и построения программного обеспечения. В настоящее время уже разработано достаточно большое количество методологий. Ниже перечислены наиболее известные:

1) модульное программирование;
2) структурное программирование;
3) логическое программирование;
4) объектно-ориентированное программирование;
5) визуальное программирование;
6) функциональное программирование;
7) концептуальное программирование;
8) доказательное программирование;
9) мобильное программирование;
10) экстремальное программирование.

Исторически, многие методологии развивались параллельно и воплощались в различных технологиях программирования и соответственно в системах программирования. Кроме того, методологии модульного, структурного, объектно-ориентированного и визуального программирования – развитие одной ветви, которая имеет универсальный характер применения. Другие, такие как концептуальное, функциональное, логическое программирование

также носят универсальный характер, но применение нашли только в области искусственного интеллекта.

Важно разграничить методологию и технологии программирования. Методология программирования больше отвечает за идейную сторону и философский аспект создания программ, в то время как технология базируется на той или иной методологии и несет больше практическую нагрузку по созданию программного обеспечения. В рамках одной методологии может быть большое количество технологий. Таким примером является методология объектно-ориентированного программирования. Технологий, поддерживающих объектно-ориентированное программирование, огромное количество. Например, OLE, COM, RUP и др

Кроме того, методологии программирования можно разделить на два класса: тяжелые и гибкие (облегченные). В первом классе разработчик воспринимается как элемент методологии, который может быть заменен. Поэтому каждый шаг в таких методологиях должен быть подробно описан. Во втором классе, человек воспринимается как существенно-нелинейный элемент методологии, а программный код является исходной документацией. В настоящее время наблюдается переход от тяжелых методологий к гибким.

Рассмотрим элементы системного анализа на примере исследования компьютерных учебных программы, которые можно отнести к сложным программным системам. Это объясняется тем, что, во-первых, учебный процесс слабо формализуем, во-вторых, сама предметная область может быть достаточно сложной для обучения и соответственно для реализации ее модели в компьютерной учебной программе, в-третьих, сложность может представлять направленность КУП для определенной группы обучаемых.

Известно, что для сложных программных систем жизненный цикл можно представить в виде шести этапов:

1. Выявление и анализ требований, предъявляемых к компьютерным учебным программам.
2. Определение спецификаций.
3. Проектирование.
4. Кодирование.
5. Тестирование и отладка.
6. Эксплуатация и сопровождение.

Рассмотрим каждый этап этого цикла.

Выявление и анализ требований, как правило, производится с помощью системного анализа. Возьмем за основу методику системного анализа, представленную в работе.

Эта методика первоначально предполагает выявление всех «заинтересованных сторон» – участников проблемной ситуации. На рисунке 2.1 представлены основные целеполагающие системы для КУП.

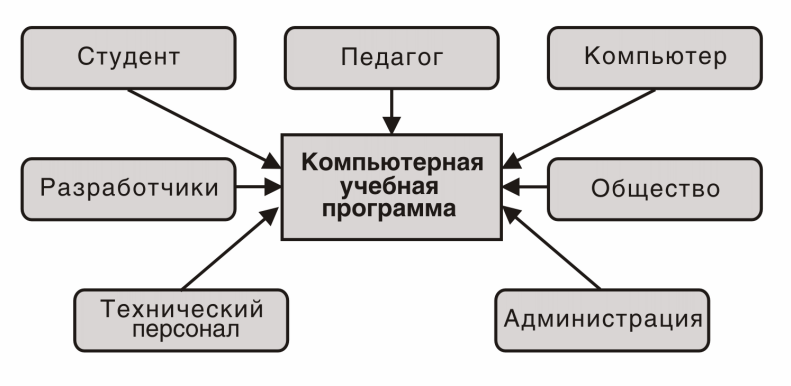

Рис. 2.1 – Основные целеполагающие системы

Это прежде всего «обучаемый» – конечный пользователь КУП. Требования со стороны обучаемого можно разделить на три группы:

- психолого-педагогические;
- инженерно-психологические;
- медицинские.

Психолого-педагогические требования в целом определяют эффективность учебного процесса. Важнейшим требованием здесь является требование «компьютерная учебная программа должна научить». При этом:

1. КУП должна адаптироваться к физиологическим и психологическим особенностям обучаемого (память, темперамент, реакция, физическое и умственное развитие, возраст, зрение, слух).

2. КУП должна быть основана на деятельностном подходе в формировании психики, эрудиции и нравственных качеств.

3. КУП должна обеспечить постоянную и положительную мотивацию деятельности обучаемого.

4. КУП должна использовать комбинированные приемы обучения, которые развивают и используют как абстрактно-логическое, так и образно-эмоциональное мышление, интуицию обучаемого.

5. КУП должна впитывать в себя последние достижения в области педагогических наук.

Инженерно-психологические требования определяют интерфейс между обучаемым и КУП. Здесь требования будут следующие:

1. Простота работы с КУП.
2. Дружелюбность интерфейса.
3. Приспособление к требованиям конкретного обучаемого, (например, настройка цвета и шрифта текста, возможность увеличения шрифта).
4. Организация комфортного интерфейса.

Медицинские требования определяют факторы КУП, которые влияют на здоровье обучаемого. Эти требования не только определяют влияние компьютера на обучаемого, но и влияние самой КУП. Прежде всего, это касается зрения, психики и нервной системы. Некоторые требования, например, время обучения с помощью некоторой КУП для разных групп учащихся, определяются федеральными санитарными правилами, нормами и гигиеническими нормативами.

Педагог непосредственно организует обучение предмета с помощью готовой КУП. Запускает при необходимости программу, наблюдает за ходом работы студента, приходит на помощь при возникновении трудностей. Регистрирует текущие успехи учащегося. Основные требования со стороны учителя следующие:

1. Обеспечение различных форм организации работы с аудиторией – от коллективной до полностью индивидуальной с каждым учащимся.
2. Обеспечение различных видов связи педагога с обучаемыми: электронная почта, доски объявлений, переадресация учащегося к учителю для личного контакта; вмешательство педагога в ход обучения на любой стадии, связь со всеми обучаемыми или с каждым в отдельности; возможность негласного контроля.
3. Различные формы накопления опыта: протоколирование процесса обучения; статистический анализ; регистрация востребуемости тех или иных разделов КУП.
4. Возможность внесения изменений в КУП (по крайней мере, адаптацию КУП для конкретного вида обучения).

Важнейшим требованием администрации является повышение эффективности процесса обучения с использованием компьютерных учебных программ. Здесь эффективность толкуется в самом общем смысле. Т.е. это может быть сокращение прямых и косвенных затрат на образование, или повышение качества обучения, или создание комфортной творческой атмосферы педагогического коллектива. Кроме указанных были выявлены следующие требования: информационное обеспечение административных функций

(сбор данных и статистический анализ, составление отчетов); соблюдение требований стандартизации и унификации.

Основным требованием со стороны технического персонала является снижение затрат на эксплуатацию разрабатываемой КУП. Это требование предусматривает:

1) простоту запуска и настройку разрабатываемой КУП;
2) минимизацию объемов требуемой памяти;
3) минимизацию времени выполнения;
4) использование стандартных технических устройств.

В процессе разработки компьютерной учебной программы выделяются три основные составляющие: психолого-педагогическая, организационно-экономическая и техническая. Психолого-педагогическая составляющая обеспечивает педагогические цели и методы достижения их в разрабатываемой КУП.

Организационно-экономическая составляющая обеспечивает реализацию и тиражирование данной КУП при заданных финансовых, трудовых и временных ограничениях. Техническая составляющая собственно реализует КУП в виде программного, информационного и иных обеспечениях.

«Разработчики» – довольно большой коллектив специалистов, состоящий из программистов, психологов, методистов, художников, музыкантов, звуковых режиссеров, мультипликаторов, сценаристов, экономистов, менеджеров по маркетингу и рекламе, юристов, медиков, специалистов по тестированию, организаторов. Условно всех разработчиков можно разделить на 11 групп (рис. 2.2).

1. Группа методистов определяет цели, содержание и этапы обучения с помощью, разрабатываемой КУП, основываясь на последних достижениях психолого-педагогической науки и практики. Определяет педагогическую технологию обучения. Эта группа формирует учебный материал для представления ее в КУП, разрабатывает контрольные вопросы и задания, определяет алгоритм оценивания знаний. В состав этой группы входят высококвалифицированные эксперты в области преподавания данного предмета (раздела знаний), обладающие большим педагогическим опытом.

2. Группа психологов обеспечивает разработку разнообразных тестов состояния здоровья, тестов способностей и достижений. Также эта группа решает вопросы разработки эффективного интерфейса между студентом и КУП.

3. Группа программистов непосредственно реализует КУП. Программистов можно разделить на системных и прикладных. Системные программисты обеспечивают интерфейс КУП с вычислительной системой конкрет-

ного класса компьютера. Прикладные программисты обеспечивают разработку отдельных модулей КУП.

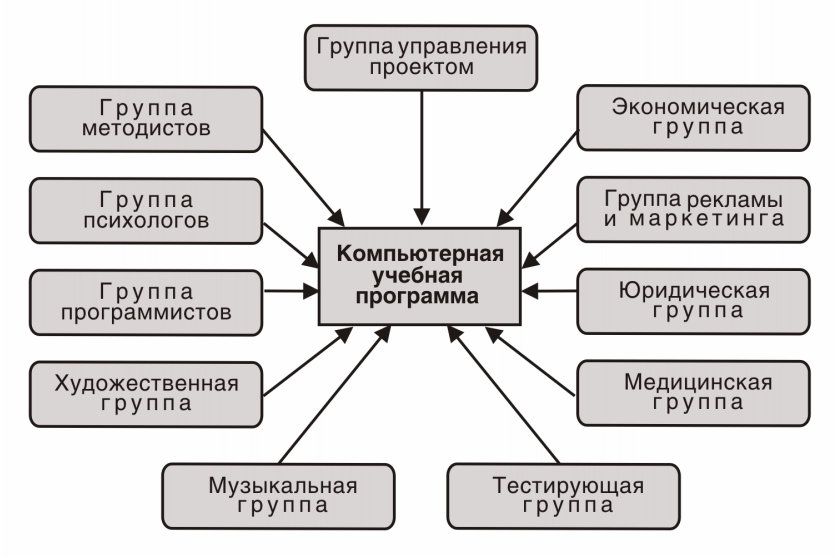

Рис. 2.2 – Основные группы разработчиков компьютерных учебных программ

4. Художественная группа обеспечивает компьютерное представление учебного материала. При этом решаются следующие задачи: выбор или разработка шрифтов для представления текстовой учебной информации. Ввод, редактирование и форматирование текста. Разработка, ввод и редактирование иллюстраций, представленных в графической форме. Создание компьютерной анимации и визуальных эффектов. Запись и редактирование видеоклипов или видеофильмов.

5. Музыкальная группа обеспечивает музыкальное сопровождение, разработку и ввод различных звуковых эффектов в КУП. Сопровождение подачи учебного материала голосом диктора.

6. Медицинская группа обеспечивает определение параметров КУП, влияющих на здоровье учащихся. Это прежде всего, предполагаемое время работы с данной КУП.

7. Экономическая группа обеспечивает экономическую обоснованность и целесообразность разработки КУП, производит поддержку финансового состояния предприятия при реализации данного проекта.

8. Группа маркетинга и рекламы обеспечивает разработку стратегии рекламной компании, разработку рекламных материалов (роликов, демонстрационных версий, рекламных листков и писем и др.), определение рынков сбыта и цен на разрабатываемую КУП, работу по заключению дилерских соглашений на тиражирование проектируемой КУП.

9. Юридическая группа обеспечивает правовую поддержку, которая включает разработку вопросов правовой защиты всех заинтересованных сторон: обучаемого, учителя, администрации и разработчиков.

10. Тестирующая группа - довольно большая группа разнообразных специалистов, производящая тестирование разрабатываемой КУП. Прежде всего, это преподаватели, имеющие опыт работы с подобными программами.

11. Группа управления проектом обеспечивает управление, планирование и координацию отдельных этапов разработки компьютерной учебной программы. В эту группу входят руководитель проекта, его помощники по отдельным направлениям, технические работники.

Развитие компьютерной техники привело к ситуации, когда на рынке компьютеров для образования осталось всего два типа. Это IBM PC–подобные и Макинтоши фирмы Apple. Основным требованием для компьютера, предназначенного для обучения, является оснащение его средствами мультимедиа. Другим важным требованием является возможность подключения компьютера в компьютерную сеть Интернет.

Еще одним важным аспектом с точки зрения реализации КУП является выбор программной платформы. В настоящее время имеется достаточно много разнообразных операционных систем: MS DOS, Windows 3.1/95/ NT/2000/2003, Unix, OS/2 и т.д. В настоящее время одной из самых распространенных операционных систем, используемых в образовании, является ОС Windows.

Следует также отметить, что развитие компьютерной техники для образовательных целей не стоит на месте и в недалеком будущем появятся принципиально новые классы компьютеров. Прообразом таких компьютеров являются компьютеры-блокноты. Главные черты новых компьютеров:

1. Основным устройством ввода будет ручка типа шариковой. Здесь компьютер будет распознавать почерк студента.

2. Экран компьютера будет расположен не вертикально, а горизонтально. Таким образом, компьютер будет лежать на столе как обычная тетрадь, в которой можно писать и рисовать.

3. В компьютере будет реализован речевой ввод и вывод информации.

4. Компьютеры будут оснащены беспроводной связью.

В целом общество также выдвигает ряд требований к разрабатываемой КУП. Это требования соблюдения безопасности использования знаний. То есть знания, полученные с помощью данной КУП, не должны использоваться во вред людям и природе. Другим важным требованием является повышение культурного уровня обучаемого. То есть обучаемый должен получать не только чистые знания по данной конкретной теме, но и выдающиеся примеры приобретения или применения этих знаний.

После выявления требований необходимо провести их анализ и записать спецификацию на разрабатываемую компьютерную учебную программу. Здесь под спецификацией будем понимать описание компьютерной учебной программы, по которому производится ее проектирование. Это описание может быть составлено на естественном языке, или может быть использован некоторый формализованный язык.

В спецификации должно быть записано: платформа и тип компьютера, определен подход к проектированию компьютерной учебной программы, определены виды представления учебного материала (например, использует ли данная КУП аудио- или видео- учебную информацию). Если в данной КУП предусматриваются тесты, то определяются тип вопроса, способы ввода ответа и его анализа, определяются способы оценивания ответов и их фиксации и т.д.

Этап проектирования предполагает разработку алгоритмов и информационной базы, для компьютерного представления учебного материала. На этом этапе готовится учебный материал, определяется его структура, пишется текст, готовятся иллюстрации, подготавливается видеоматериал, подбирается аудиоматериал. Для тестовых программ готовятся вопросы, разрабатываются алгоритмы анализа и оценивания ответов. Разрабатываются алгоритмы представления и предъявления учебной информации. Ниже будут подробно рассмотрены конкретные методы проектирования для определенного класса компьютерных учебных программ.

На этапе кодирования непосредственно создается компьютерная учебная программа. Кодирование включает создание информационной базы, состоящей из текста, иллюстраций, анимации, аудио- и видео -информации. Причем вся эта учебная информация связана в виде некоторой информационной сети. Например, отдельные фрагменты текста, иллюстраций и т.д. представлены в виде отдельных компьютерных страниц (кадров), которые в свою очередь могут быть связаны с другими кадрами.

Алгоритмы, разработанные на этапе проектирования, реализуются программно с использованием некоторой инструментальной среды.

На данном этапе производится комплексная сборка всех модулей КУП и отладка программы.

Тестирование является необходимым и важным этапом создания КУП. На данном этапе производится проверка функционирования всех составных частей КУП, определяются реальные характеристики. Первоначально тестируются отдельные модули программы. Далее производится тестирование информационного обеспечения. Производится проверка представления и предъявления информации с точки зрения обучения данному предмету, производится определение корректности вопросов и способов их оценивания. Проверяется связанность учебной информации (например, все ссылки должны быть на реальные кадры). Далее производится проверка режима ожидания ввода, т.е. всех ситуаций ввода информации со стороны обучаемого.

Порядок проведения занятия

1. Выбрать одну задач из области паспорта специальности 05.13.11 или тему диссертационного исследования
2. Провести системны анализ. Построить схему целеполагающих систем, существенно влияющих на решение данной задачи
3. Провести анализ каждой из них
4. Записать список требований для каждой целеполагающей системы.
5. Оценить значение требований.
6. Записать список требований в виде технического задания.
7. В отчете представить ход построения технического задания.

Рекомендуемые источники

1. Силич М.П., Силич В.А., Основы теории систем и системного анализа: учеб. пособие. – Томск: Изд-во Томск. гос. ун-та систем управления и радиоэлектроники, 2013. – 340 с.
2. Кручинин В.В. Разработка компьютерных учебных программ – Томск:Изд-во ТГУ, 1998. – 212с.
3. Гарайс Д. В. Новые технологии в программировании: Учебное пособие [Электронный ресурс] / Д. В. Гарайс, А. Е. Горяинов, А. А. Калентьев – Томск: ТУСУР, 2014. – 176 с. – Режим доступа: https://edu.tusur.ru/publications/5796 (дата обращения: 15.05.2018).
4. Салмина Н. Ю., Функциональное программирование и интеллектуальные системы: учебное пособие [Электронный ресурс] / Салмина Н. Ю. – Томск: ТУСУР, 2016. – 100 с. – Режим доступа: https://edu.tusur.ru/publications/6357 (дата обращения: 15.05.2018).
5. Панов, С. А. Теория и технологии программирования: Курс лекций [Электронный ресурс] / С. А. Панов. – Томск: ТУСУР, 2015. – 116 с. – Режим доступа: https://edu.tusur.ru/publications/5013 (дата обращения: 15.05.2018).
6. Кручинин, В. В. Технологии программирования: Учебное

пособие [Электронный ресурс] / В. В. Кручинин. – Томск: ТУСУР, 2013. – 271 с. – Режим доступа: https://edu.tusur.ru/publications/2834 (дата обращения: 15.05.2018).

7. Перегудов Ф.И., Тарасенко Ф.П. Основы системного анализа: Учеб. пособие. – 3-е изд. – Томск: Изд-во НТЛ, 2001. – 396 с.

2.2 Практическое занятие «Методика выбора инструментальных систем программирования (ИС)»

Порядок проведения занятия

1. Выбрать или разработать методику сравнения инструментальных систем с учетом решаемой задачи
2. Разработать систему критериев и множества показателей инструментальных систем
3. Разработать процедуру определения значений показателей для конкретной ИС
4. Выбрать множество инструментальных систем
5. Записать таблицу значений показателей для выбранных ИС.
6. На основе использования критериев выбрать наиболее подходящую ИС
7. Записать достоинства и недостатки выбранной ИС

Рекомендуемые источники

1. Информатика : учебник для вузов / Н. В. Макарова, В. Б. Волков. – СПб. : ПИТЕР, 2012. – 576 с.

2. Информатика: базовый курс : учебник для вузов / О. А. Акулов, Н. В. Медведев. - 8-е изд., стереотип. – М. : Омега-Л, 2013. – 576 с.

3. Информатика. Базовый курс : Учебник для вузов / С. В. Симонович [и др.] ; ред. : С. В. Симонович. – 2-е изд. – СПб. : Питер, 2007. – 639[1] с.

4. C/C++. Программирование на языке высокого уровня : учебник для вузов / Т. А. Павловская. – СПб. : ПИТЕР, 2013. – 461 с.

5. Кручинин, В. В. Технологии программирования: Учебное пособие [Электронный ресурс] / В. В. Кручинин. – Томск: ТУСУР, 2013. – 271 с. – Режим доступа: https://edu.tusur.ru/publications/2834 (дата обращения: 15.05.2018).

2.3 Практическое занятие «Методы выбора систем баз данных и знаний»

Порядок проведения занятия

1. Выбрать или разработать методику сравнения систем управления базами данных (СУБД) с учетом решаемой задачи.

2. Разработать систему критериев и множества показателей СУБД).
3. Разработать процедуру определения значений показателей для конкретной СУБД.
4. Выбрать множество подходящих СУБД.
5. Записать таблицу значений показателей для выбранных СУБД.
6. На основе использования критериев выбрать наиболее подходящую СУБД.
7. Записать достоинства и недостатки выбранной СУБД.

Рекомендуемые источники

1. Управление данными : учебник для вузов / А. В. Кузовкин, А. А. Цыганов, Б. А. Щукин. – М. : Академия, 2010. – 256 с.
2. Салмина Н. Ю., Функциональное программирование и интеллектуальные системы: учебное пособие [Электронный ресурс] / Салмина Н. Ю. – Томск: ТУСУР, 2016. – 100 с. – Режим доступа: https://edu.tusur.ru/publications/6357 (дата обращения: 15.05.2018).
3. Представление знаний в информационных системах [Текст] : учебник для вузов / Б. Я. Советов, В. В. Цехановский, В. Д. Чертовской. – М. : Академия, 2011. – 144 с. : табл. – (Высшее профессиональное образование. Информатика и вычислительная техника). – Библиогр.: с. 140-142.

2.4 Практическое занятие «Основные функции системы программирования для систем компьютерной алгебры»

Порядок проведения занятия

1. Установить полную версию системы компьютерной алгебры Maxima.
2. Разобрать общий механизм реализации системы компьютерной алгебры.
3. Изучить основные функции символьных вычислений: дифференцирование, интегрирование, преобразование выражений, разложение функции в ряд, решение уравнений.
4. Рассмотреть основные операторы встроенного языка программирования.
5. Реализовать один из алгоритмов.
6. Рассмотреть возможности системы для интеграции.

Рекомендуемые источники

1. Кручинин, В. В. Профессиональные математические пакеты: Учебно-методическое пособие [Электронный ресурс] / Кручинин В. В., Перминова М. Ю. – Томск: ТУСУР, 2017. – 117 с. – Режим доступа: https://edu.tusur.ru/publications/7256 (дата обращения: 15.05.2018).

2.5 Практическое занятие «Применение методов анализа алгоритмов, вычислительные эксперименты»

Порядок проведения занятия

1. Изучить методы и инструментальные средства анализа алгоритмов.
2. Выбрать один из алгоритмов.
3. Разработать схему экспериментального исследования алгоритма.
4. Разработать программу и отладить ее.
5. Построить графики вычислительной сложности алгоритма в худшем случае, в среднем случае.

Рекомендуемые источники

1. Ульянов, М.В. Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ [Электронный ресурс] : учеб. пособие – Электрон. дан. – М. : Физматлит, 2008. – 304 с. – Режим доступа: https://e.lanbook.com/book/2354 (дата обращения: 15.05.2018).

2.6 Практическое занятие «Методы грубой силы»

Теоретические основы

Примеры задач, решающееся методом полного перебора.

1. Сортировка. Составьте n! перестановок данных элементов и просмотрите их пока не будет найдена искомая перестановка.
2. Задача n ферзей. Расставить n ферзей на шахматной доске размера n × n так, что никакие два из них не стоят в одном ряду, одной колонке или на одной диагонали. В разных рядах и разных колонках − n! Вариантов
3. Числовые ребусы. Заменить буквы на цифры так, чтобы выполнялось данное равенство, например, SEND + MORE = MONEY.

Достоинства: применим к широкому кругу дискретных задач; простота; трудно или невозможно для некоторых задач единственный способ решения: умножение матриц; NP-сложные задачи (задача коммивояжёра, задача о рюкзаке, и сотни других задач дискретной оптимизации).

Недостатки: Метод эффективен для малой размерности задачи.

Порядок проведения занятия

1. Изучить достоинства и недостатки методов полного перебора.
2. Построить алгоритм перебора множества перестановок.
3. Разработать и отладить программу.
4. Построить схему вычислительного эксперимента.
5. Представить алгоритм, программу и график вычислительной

сложности в отчет.

Рекомендуемые источники

1. Ульянов М.В. Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ [Электронный ресурс] : учеб. пособие – Электрон. дан. – М. : Физматлит, 2008. – 304 с. – Режим доступа: https://e.lanbook.com/book/2354 (дата обращения: 15.05.2018).

2.7 Практическое занятие «Метод ветвей и границ»

Теоретические основы

Метод ограниченного перебора, в котором обеспечивается сокращение вариантов перебора за счет отбрасывания частично построенных кандидатов, которые не могут привести к правильному решению задачи. Этот метод сопряжен с построением дерева решений, которое строится по следующему алгоритму:

- вершины: частичные решения;
- рёбра: переходы к расширенным частичным решениям;
- порядок построения: дерево решений строится на манер алгоритма «поиск в глубину»;
- отбрасывание бесперспективных вершин: не рассматривайте ветви дерева, которые идут из бесперспективных вершин – возвращайтесь к родительской вершине.

Порядок проведения занятия

1. Изучить достоинства и недостатки методов полного перебора
2. Построить алгоритм перебора расстановки шахматных фигур на доске (nxn) (ферзей, ладей, коней, слонов).
3. Разработать и отладить программу
4. Построить схему вычислительного эксперимента
5. Представить алгоритм, программу и график вычислительной сложности в отчет

Рекомендуемые источники

1. Ульянов, М.В. Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ [Электронный ресурс] : учеб. пособие – Электрон. дан. – Москва : Физматлит, 2008. – 304 с. – Режим доступа: https://e.lanbook.com/book/2354. (дата обращения: 15.05.2018).
2. Кремер Н.Ш. и др. Исследование операций в экономике. Учебное пособие для вузов/ ред. : Н. Ш. Кремер. – М. : ЮНИТИ, 2006. – 407 с.
3. Грибанова, Е. Б. Исследование операций и методы оптимизации : Учебное пособие [Электронный ресурс] / Грибанова Е. Б., Мицель А. А. –

Томск: ТУСУР, 2017. – 185 с. – Режим доступа: https://edu.tusur.ru/publications/7127 (дата обращения: 15.05.2018).

2.8 Практическое занятие «Метод «Разделяй и властвуй»»

Теоретические основы

Этот метод можно определить как метод, который рекурсивно делит задачу на несколько подзадач, до тех пор пока они становятся достаточно простыми для прямого решения. Затем эти решения подзадач объединяются, чтобы получить решение первоначальной задачи.

Примеры задач:

1) сортировка слиянием и быстрая сортировка;
2) метод Карацубы для умножения больших чисел;
3) умножение матриц методом Штрассена;
4) обход бинарного дерева и родственные алгоритмы;
5) декомпозиционные методы для нахождения ближайшей пары точек и построения выпуклой оболочки;
6) быстрое преобразование Фурье.

Порядок проведения занятия

1. Изучить достоинства и недостатки метода декомпозиции.
2. Построить один из предложенных выше алгоритм или алгоритм, связанный с темой исследования.
3. Разработать и отладить программу.
4. Построить схему вычислительного эксперимента.
5. Представить алгоритм, программу и график вычислительной сложности в отчет.

Рекомендуемые источники

1. Ульянов М.В. Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ [Электронный ресурс] : учеб. пособие – Электрон. дан. – Москва : Физматлит, 2008. – 304 с. – Режим доступа: https://e.lanbook.com/book/2354 (дата обращения: 15.05.2018).

2.9 Практическое занятие «Жадный метод»

Теоретические основы

Жадный метод – это подход к решению задач оптимизации, в котором на каждом шагу делается выбор, который:

1) удовлетворяет всем ограничениям задачи;
2) локально оптимален;
3) необратим.

Жадный подход приводит к корректному решению некоторых задач оптимизации, но не работает для других.

Примеры жадных алгоритмов

1) задача о размене;
2) алгоритмы Прима и Крускала для нахождения минимального остовного дерева (МОД);
3) алгоритм Дейкстры;
4) коды Хаффмана;
5) приближённые алгоритмы для задач комбинаторной оптимизации (например, правило ближайшего соседа в задаче коммивояжёра);
6) градиентные методы спуска для минимизации функций.

Порядок проведения занятия

1. Изучить достоинства и недостатки метода
2. Построить один из предложенных выше алгоритм или алгоритм, связанный с темой исследования
3. Разработать и отладить программу
4. Построить схему вычислительного эксперимента
5. Представить алгоритм, программу и график вычислительной сложности в отчет

Рекомендуемые источники

1. Кремер Н.Ш. и др. Исследование операций в экономике. Учебное пособие для вузов/ ред. : Н. Ш. Кремер. – М. : ЮНИТИ, 2006. – 407 с.
2. Грибанова, Е. Б. Исследование операций и методы оптимизации : Учебное пособие [Электронный ресурс] / Грибанова Е. Б., Мицель А. А. – Томск: ТУСУР, 2017. – 185 с. – Режим доступа: https://edu.tusur.ru/publications/7127 (дата обращения: 15.05.2018).
3. Ульянов, М.В. Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ [Электронный ресурс] : учеб. пособие – Электрон. дан. – М. : Физматлит, 2008. – 304 с. – Режим доступа: https://e.lanbook.com/book/2354 (дата обращения: 15.05.2018).

2.10 Практическое занятие «Метод динамического программирования»

Теоретические основы

В теории оптимизации этот подход интерпретируется как метод решения оптимизационных задач, которые удовлетворяют принципу оптимальности. (Ричард Беллман)

В информатике динамическое программирование интерпретируется как подход к решению задач с пересекающимися подзадачами, не обязательно оптимизационного характера:

- решите все меньшие подзадачи, но один раз;
- занесите решения в таблицу;
- возьмите решение для нужного случая из этой таблицы.

Примеры задач

1. Вычисление чисел Фибоначчи и биномиальных коэффициентов.
2. Оптимальное умножение нескольких матриц.
3. Построение оптимального бинарного дерева поиска.
4. Алгоритм Воршалла для транзитивного замыкания.
5. Алгоритм Флойда для поиска кратчайших путей между всеми парами вершин взвешенного графа.
6. Некоторые трудные задачи дискретной оптимизации (например, задача о рюкзаке).

Порядок проведения занятия

1. Изучить достоинства и недостатки метода.
2. Построить один из предложенных выше алгоритм или алгоритм, связанный с темой исследования.
3. Разработать и отладить программу.
4. Построить схему вычислительного эксперимента.
5. Представить алгоритм, программу и график вычислительной сложности в отчет.

Рекомендуемые источники

1. Кремер Н.Ш. и др. Исследование операций в экономике. Учебное пособие для вузов/ ред. : Н. Ш. Кремер. – М. : ЮНИТИ, 2006. – 407 с.
2. Грибанова, Е. Б. Исследование операций и методы оптимизации : Учебное пособие [Электронный ресурс] / Грибанова Е. Б., Мицель А. А. – Томск: ТУСУР, 2017. – 185 с. – Режим доступа: https://edu.tusur.ru/publications/7127 (дата обращения: 15.05.2018).
3. Ульянов М.В. Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ [Электронный ресурс] : учеб. пособие. – Электрон. дан. – М. : Физматлит, 2008. – 304 с. – Режим доступа: https://e.lanbook.com/book/2354 (дата обращения: 15.05.2018).

2.11 Практическое занятие «Методы комбинаторной генерации, основанные на деревьях И/ИЛИ»

Теоретические основы

Развитие исследований в области комбинаторики и методов программирования привело к понятию комбинаторной генерации, где исследуются алгоритмы генерации и нумерации комбинаторных объектов. Здесь под комбинаторными объектами понимается конструктивные объекты, которые имеют финитную природу и получены по конечным правилам для некоторого конечного числа конечных множеств. К таким комбинаторным объектам относятся сочетания, перестановки, размещения, разбиения, разложения, деревья, графы, контекстно-свободные грамматики и т.д.

Исторический обзор развития этого направления дает Д.Кнут в 4 томе «Искусство программирования». Однако, классификацию алгоритмов генерации и определение понятий не приводит. Такую классификацию приводит Ф.Раски в работе «Комбинаторная генерация». Он выделяет 4 класса алгоритмов генерации:

1) последовательная генерация множества всех комбинаторных объектов данного класса (listing);

2) нумерация всех комбинаторных объектов данного класса (ranking);

3) генерация комбинаторных объектов в соответствии с процессом нумерации (unranking);

4) случайная генерация реализации комбинаторного объекта (random selection).

Алгоритмы последовательной генерации комбинаторных объектов наиболее изучены. Общая схема их следующая:

1) установка начального комбинаторного объекта;

2) организация цикла, в котором производится вывод следующего комбинаторного объекта или его конструирование из данного.

Этот процесс можно реализовать двумя способами. Первый способ предполагает реализацию процедуры генерации как контейнера, т.е. для всех объектов данного класса вызывается некоторая заданная процедура. Обычно такой механизм реализуется в виде процедуры под названием *ForEach*.

Второй способ подразумевает реализацию процедур генерации *First* и *Next*. Процедура *First* обеспечивает установку начального комбинаторного объекта. Процедура *Next* генерирует следующий объект.

Нумерация рассматривается как процесс упорядочения множества комбинаторных объектов. Если некоторое множество комбинаторных объектов упорядочено, то номер позиции некоторого объекта в этом упорядочении будет являться порядковым номером (*rank*). Соответствующий алгоритм или процедуру обычно называют *Rank*.

Процесс генерации комбинаторного объекта (*Unranking*) по его номеру (рангу) является обратным процессу нумерации. Соответствующий алгоритм или процедуру называют *Generate*.

В некоторых случаях желательно наличие алгоритма генерации со случайным выбором (*random selection*), когда последовательность объектов не упорядочена или не требуется все множество комбинаторных объектов.

В общем случае, для получения процедур генерации *Generate* и нумерации *Rank* множества необходимо задать биективное отображение:

$$Gen: N_n \to A_n,$$

где N_n – конечное множество натуральных чисел;

A_n – множество объектов данного комбинаторного класса.

Тогда обратное отображение $Rank \equiv Gen^{-1}$ задает процедуру нумерации:

$$Rank: A_n \to N_n.$$

На рисунке 2.3 показаны основные элементы и схема взаимодействия алгоритмов *Generate* и *Rank*. Алгоритм *Generate* на основании числа $num \in N_n$ и некоторого описания D создает элемент $a \in A_n$. Алгоритм *Rank* производит обратное действие, на основании $a \in A_n$ и некоторого описания C, формирует номер $num \in N_n$.

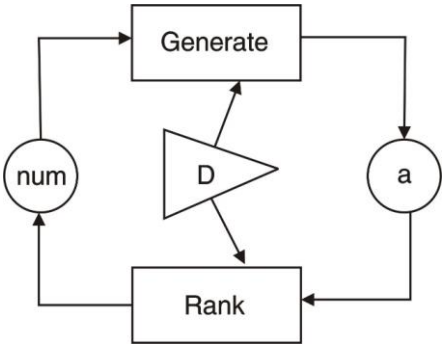

Рис 2.3 – Основная схема алгоритмов *Generate* и *Rank*

В работе [1] рассмотрены методы построения алгоритмов генерации и нумерации комбинаторных объектов, основанные на представлении описания *D* в виде дерева И/ИЛИ.

Основные результаты:

1. Введено понятие варианта дерева И/ИЛИ и правила построения дерева И/ИЛИ для представления множества. Доказано, что если для некоторого множества получено дерево И/ИЛИ, то каждому элементу множества взаимно-однозначно соответствует вариант дерева И/ИЛИ. Таким образом, мощность множества равна числу вариантов в дереве И/ИЛИ.

2. Показано, что если мощность множества комбинаторных объектов задана рекурсивной функцией, сводимой к примитивно-рекурсивной, алгебры $\{N,+,\times,R,S\}$, то для такого множества существует схема рекурсивной композиции построения дерева И/ИЛИ.

3. Показано, что если для некоторого множества комбинаторных объектов получена схема рекурсивной композиции построения дерева И/ИЛИ, то существует рекурсивная функция, сводимая к примитивно-рекурсивной, алгебры $\{N,+,\times,R,S\}$, вычисляющая мощность данного множества.

4. Получены алгоритмы *Generate* и *Rank*, обеспечивающие биективное отображение $Gen: N_n \to W(D)$ между конечным множеством натуральных чисел N_n и множеством вариантов дерева И/ИЛИ $W(D)$. Исследованы их свойства.

5. Предложены методы построения алгоритмов генерации и нумерации множества комбинаторных объектов, основанные на представлении этого множества в виде дерева И/ИЛИ.

6. Предложенные методы применены для построения алгоритмов генерации и нумерации объектов следующих комбинаторных объектов: сочетаний, перестановок, размещений, разбиений, беспорядков, множеств Фибоначчи и Сильвестра; деревьев (двоичных, АВЛ, полных двоичных, 2-3 деревьев и т.д.); решений переборных задач и выражений языков, заданных контекстно-свободными грамматиками.

Порядок проведения занятия

1. Изучить понятие деревьев И/ИЛИ.
2. Изучить операцию рекурсивной композиции деревьев.
3. Изучить метод построения алгоритмов последовательной генерации.
4. Построить схему рекурсивной композиции деревьев И/ИЛИ.
5. Построить автомат последовательной генерации.
6. Построить биекцию между деревом и комбинаторным множеством.

2.12 Практическое занятие «Алгоритмы комбинаторной генерации сочетаний, разбиений и композиций»

Теоретические основы

Алгоритмы генерации и нумерации сочетаний

Сочетанием элементов из $E = \{a_1, a_2, ..., a_n\}$ по m называется упорядоченное подмножество из m элементов, принадлежащих E и отличающихся друг от друга составом, но не порядком элементов. Число сочетаний вычисляется по формуле

$$C_n^m = \frac{n!}{(n-m)!m!}.$$

Сочетание можно представить двоичным числом, где n- число разрядов в числе, m- число единиц в n-разрядном двоичном числе. Тогда наличие единицы в i-разряде будет означать, что элемент a_i войдет в сочетание.

Построение дерева И/ИЛИ. Существует множество различных алгоритмов генерации сочетаний ниже описан алгоритм генерации, основанный на представлении сочетания в виде дерева И/ИЛИ. Рас-

смотрим построение дерева И/ИЛИ для сочетания C_n^m. Для этого запишем следующее тождество:

$$C_n^m = C_{n-1}^m + C_{n-1}^{m-1}.$$

Применяя данное тождество для заданного сочетания C_n^m, строится схема рекурсивной композиции (рис. 2.4).

Дерево строится до тех пор, пока каждый лист этого дерева не станет элементом тождества C_k^0 или C_k^k, для которых $C_k^0 = C_k^k = 1$. Очевидно, что число листьев такого дерева равно C_n^m. Дерево является двоичным, т.к. каждый узел, кроме листьев, имеет ровно два сына.

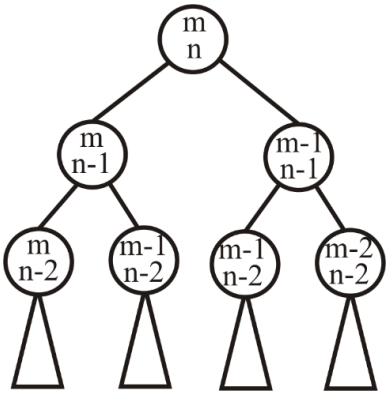

Рис.2.4 – Схема рекурсивной композиции для представления сочетания

Рассмотрим данное двоичное дерево как дерево И/ИЛИ. Все узлы этого дерева являются ИЛИ-узлами, тогда каждый вариант включает путь от корня дерева к листу и число таких вариантов будет равно числу сочетаний. Функция $\omega(z) = C_n^m$, где *n* и *m*- параметры сочетания, записанные в узле *z*. Теперь, если в каждый лист такого дерева запишем конкретное сочетание, то можно использовать такое дерево для генерации сочетаний. Однако, можно предложить более экономичный алгоритм, поскольку для узла *z(n,m)* можно найти $\omega(s_1) = C_{n-1}^m$ и $\omega(s_2) = C_{n-1}^{m-1}$.

Таким образом, число вариантов и листьев в дереве И/ИЛИ для представления сочетаний равно C_n^m, плотность вариантов на число листьев равна 1. Вариант представляет собой путь от корня к листу.

Построение отображения варианта в объект. Если дуги нагрузить числами, левая -0, правая- 1, то путь от корня к листу идентифицирует некоторое двоичное число, это двоичное число будет соответствовать некоторому сочетанию. Пример такого дерева записан для представления сочетания C_4^2 на рисунке 2.5. Число листьев равно 6, что соответствует числу сочетаний. Заметим, что число путей из корня этого дерева до листьев равно числу сочетаний. Тогда, произведя левосторонний обход дерева (рис. 2.5) и записав соответствующие цифры для каждого пути, получим первый столбец таблицы 2.1.

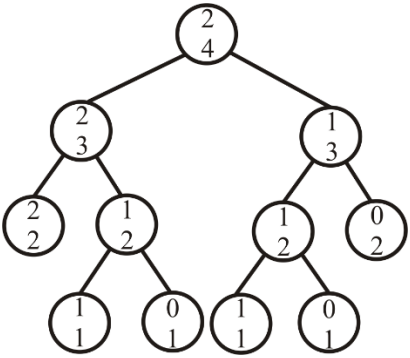

Рис. 2.5 – Пример дерева И/ИЛИ для сочетания C_4^2

Таблица 2.1

00	(11)
010	(1)
011	(0)
100	(1)
101	(0)
11	(00)

Для первой строчки не хватает двух единиц, для второй – одной единицы, для третьей – нуля, для четвертой – единицы, для пятой – нуля, для шестой – двух нулей. Можно предложить следующее правило дописывания единиц или нулей: если лист содержит *m* равное *n*,

то справа дописывается m единиц; если лист содержит *m* равное нулю, то справа дописывается *n* нулей.

Используя эти простые правила, можно перечислить все сочетания для заданных значений *m* и *n*.

Алгоритм генерации сочетаний

Построим алгоритм генерации сочетания по трем параметрам, где *m, n* – параметры сочетания, а *num* – номер сочетания. Очевидно, что для *k* должно выполняться следующее соотношение:

$$1 \le num \le C_n^m.$$

Решающее правило будет следующим:

1. Если $num \le C_{n-1}^m$, то записывается ноль и происходит переход на рассмотрение узла C_{n-1}^m.

2. Если $num > C_{n-1}^m$, то записывается единица, $num = num - C_{n-1}^m$ и происходит переход на рассмотрение узла C_{n-1}^{m-1};

3. Если *n* равно *m*, то дописывается *m* нулей.

4. Если *m* равно 0, то дописываем *n* единиц.

Тогда алгоритм будет следующий:

```
algorithm GCombination (num, m, n)
 begin
  if m=0 then  { записать n нулей }
   for (i=1,n) v ← 0
    else
    if m=n  then  { записать m единиц}
     for(i=1,m) v ← 1
   else
    if num ≤ C(n-1)^m then { движение по левой ветви}
     begin
      v ← 0
     GCombination(num, m, n-1)
      end
    else
```

begin {*движение по правой ветви*}

$v \leftarrow 1$

GCombination($num - C_{n-1}^{m}$*, m-1 ,n-1)*

end

end

Здесь *num* – номер сочетания, *m, n* – параметры сочетания, *v* – вектор, содержащий двоичное представление сочетания. Операция $v \leftarrow b$ означает дописывание справа значение бита *b* в двоичном числе *v*. Алгоритм является рекурсивным. Глубина рекурсии зависит от глубины И/ИЛИ дерева сочетаний и равна параметру *n*, следовательно, сложность данного алгоритма равна *O(n).*

Алгоритм нумерации сочетания

1. Строится вариант по сочетанию, значения битов показывают направления движения по дереву И/ИЛИ.

2. В обратном порядке находится значение номера сочетания.

algorithm *RankCombination (V,k,m,n)*

begin

if *m=0* **then return** 0; {достигли листа}

if *m=n* **then return** 0**;** {достигли листа}

if V[k]=0 **then** { *движение по левой ветви*}

return *RankCombination(V,k+1,m,n-1)*

else {*движение по правой ветви*}

return *RankCombination(V,k+1,m-1,n-1)+* C_{n-1}^{m}

end

Порядок проведения занятия

1. Построить схему рекурсивной композиции деревьев И/ИЛИ на основе рекуррентной формуле для сочетаний, композиций, разбиений, чисел Фибоначчи
2. Построить автомат последовательной генерации
3. Построить биекцию между деревом и комбинаторным множеством
4. Построить алгоритмы комбинаторной генерации: нумерации, генерации по номеру, последовательной генерации
5. Реализовать функции, отладить.
6. Провести исследование полученных алгоритмов.

2.13 Практическое занятие «Алгоритмы комбинаторной генерации множеств, заданных числами Каталана»

Теоретические основы

Дадим ряд определений:

1. *Функция (отображение, оператор, преобразование)* – математическое понятие, отражающее однозначную парную связь элементов одного множества с элементами другого множества, или, функция – это правило, по которому каждому элементу одного множества, называемого областью задания (или областью определения) функции, ставится в соответствие некоторый элемент другого множества, называемого областью значений.

2. *Биекция* – это отображение, при котором каждому элементу одного множества соответствует ровно один элемент другого множества. *Биективное отображение* называют взаимно-однозначным отображением (соответствием). Для биективного отображения определено обратное отображение, которое также биективно.

3. *Решетчатый путь* – это путь в дискретном пространстве, представленного некоторой решеткой. Простой пример решетки, листок школьной тетради, разлинованный в клетку. Каждое i-ое пересечение, имеет координаты (x_i,y_i) Путь строится из некоторого множества шагов. Задача перечисления путей на решетках имеет следующий вид. Пусть имеется начальная (0,0) и конечная точки (n,m) и задано фиксированное множество видов шагов (например, шаг влево и шаг вверх). Найти все пути из начальной точки в конечную используя шаги и заданного множества.

4. *Путь Дика* – это решетчатый путь из точки (0,0) в точку (n,n), используя шаги (0,1) и (1,1) или это решетчатый путь из точки (0,0) в точку (0,2n) используя шаги (1,1) и (1,-1).

5. *Путь Моцкина* – это решетчатый путь из точки (0,0) в точку (n,0), с шагами вида (1,0),(1,1) и (1,-1), не опускающиеся ниже оси x.

6. *Путь Шредера* – это решетчатый путь из точки (0,0) в точку (n,n), с шагами вида (1,0),(0,1) и (1,1), не пересекающие линию y=x.

7. *Числом Каталана* называется число различных правильных скобочных структур из n пар скобок. Например, все правильные скобочные структуры из 3 пар скобок:

()()(),~~ ()(()),~~ (()()),~~ (())(),~~ ((())).

Первые 11 членов последовательности чисел Каталана (последовательность A000108\ в онлайн-энциклопедии целых последовательностей) следующая:

1,~~ 1,~~ 2,~~ 5,~~ 14,~~ 42,~~ 132,~~ 429,~~ 1430,~~ 4862,~~ 16796

Имеется выражение для чисел Каталана, записанное в явном виде:

$$C_n = \frac{1}{n+1}\binom{2n}{n}$$

Существует рекуррентное соотношение:

$$C_{n+1} = \sum_{k=0}^{n} C_k C_{n-k}.$$

Это рекуррентное соотношение позволяет построить следующую рекурсивную композицию, представленную на рисунке 2.6.

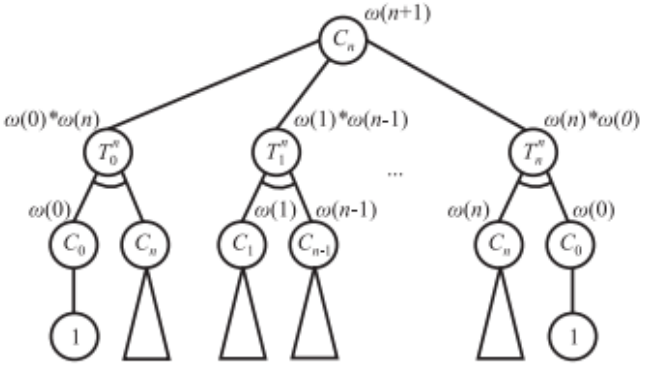

Рис 2.6 – Схема рекурсивной композиции для чисел Каталана

Тогда алгоритм генерации скобочной записи будет следующий:

```
1  algorithm GenCatalan (num,n) begin
2  |  if n = 0 then return
3  |
4  |  S:=0
5  |  for (i := 0 to n − 1) do
6  |  |  if num < S + C_i · C_{n−1−i} then {определяем T_i^n}
7  |  |  |  num:=num-S {вычисляем l1 и l2 для узлов C_i и C_{n−i} }
8  |  |  |  l1 := num  mod C_i
9  |  |  |  print("(") { печатаем открывающую скобку }
10 |  |  |  GenCatalan (l1,i) { совершаем рекурсивный спуск для левого сына}
11 |  |  |  print(")") { печатаем закрывающую скобку }
12 |  |  |  l2 := num/C_i
13 |  |  |  GenCatalan (l2, n − 1 − i) { совершаем рекурсивный спуск для правого
   |  |  |   сына}
14 |  |  |  return
15 |  |  end
16 |  |  S := S + C_i · C_{n−1−i}
17 |  end
18 end
```

Ниже представлен алгоритм нумерации скобочной записи

```
1  algorithm RankCatalan (n) begin
2  |  poso :=match("(")
3  |  if poso=-1 then  return 0
4  |
5  |  l1:=RankCatalan (n-1)
6  |  posz :=match(")")
7  |  k := (posz − poso − 1)/2
8  |  l2:=RankCatalan (n-1)
9  |  S := 0
10 |  for (i := 0 to k − 1) do  S := S + Ci * Ck−i+1
11 |
12 |  return S + l1 + l2 · Ck
13 end
```

Варианты заданий

1. Бинарные расстановки скобок в строке, состоящей из n+1 букв. Ниже приведен пример все расстановки скобок в строке xxxx

(x(x(xx)))

(x((xx)x))

((xx)(xx))

((x(xx))x)

(((xx)x)x)

Указание. Необходимо модифицировать алгоритм GenCatalan. При n=0 печатать x. Перед первым вызовом GenCatalan печатать скобку открывающую. После второго вызова GenCatalan печатать закрывающую скобку.

2. Двоичные деревья с n вершинами.

Указание: Получить скобочною строку с помощью алгоритма GenCatalan, затем заменить открывающую скобку на 01, и закрывающую скобку на 10

В полученной бинарной строке убрать первый и последние символы (0 или 1). Затем использовать рекурсивный алгоритм генерации двоичного дерева.

вход: строка C содержащая последоватеьность

нулей или единиц

выход: двоичное дерево

Создать корень и сделать его текущим

Генерация(С):

получить два бита ab из стоки С, и удалить их из С

если ab = 01 то записать новый узел к текущему как левый,
сделать его текущим.
вызывать Генерация(с)

если ab = 10 то записать новый узел к текущему как правый,
сделать его текущим.
вызывать Генерация(с)

если ab = 00 то занести в стек текущий узел,
записать новый узел к
текущему как левый, сделать его текущим.
вызывать Генерация(с)

если ab = 11 то взять из стека текущий узел,
записать новый узел к
текущему как правый, сделать его текущим.
вызывать Генерация(с)

конец

3. Полные двоичные деревья с 2(n+1) вершинами или n+1 концевыми вершинами.

Полным двоичным деревом называется дерево, у которого узлы содержат ровно два сына. Ниже представлены все полные двоичные деревья с 4 концевыми вершинами (или с семью вершинами). Скобки () обозначают узел. Итого 4 листа и 3 узла, всего 7 вершин.

(()(()(()())))
(()((()())()))
((()())(()()))
((()(()()))())
(((()())())())

Указание. Необходимо модифицировать алгоритм GenCatalan. При входном перед каждым вызовом GenCatalan печатать скобку открывающую. После каждого вызова GenCatalan печатать закрывающую скобку. Преобразовать поученную скобочную запись в полное двоичное дерево.

4. Плоские деревья с n+1 вершинами.

Указание: Получить скобочное представление, используя алгоритм GenCatalan. Создать узел, к этому узлу присоединить все узлы, порождаемые парой скобок (T) первого уровня и тд

5. Решетчатые пути из точки (0,0) в точку (n,n) с шагами вида (0,1) или (1,0), не поднимающихся выше прямой y=x.

Биекция: открывающая скобка соответствует шагу (1,0) (шаг вправо), закрывающая скобка соответствует шагу (0,1) (шаг вверх)

6. Пути Дика из точки (0,0) в точку (2n,0), т.е. решетчатые пути с шагами вида (1,1) и (1,-1), не опускающиеся ниже оси x.

Биекция: открывающая скобка соответствует шагу (1,1) (шаг вверх по диагонали), закрывающая скобка соответствует шагу (1,-1) (шаг вниз по диагонали)

7. Пути Дика из точки (0,0) в точку (2n+2,0), такие, что любая максимальная последовательность соседних шагов вида (1,-1), заканчивающаяся на оси x, имеют нечетную длину

8. Пути Дика (определение см. в предыдущем пункте) из точки (0,0) в точку (2n+2,0), не имеющие пиков на высоте два.

9. Неупорядоченные пары решетчатых путей, состоящих их $n+1$ шагов, которые начинаются в точке (0,0), имеют шаги вида (1,0) или (0,1), заканчиваются в одной и той же точке и пересекаются только в начальной и конечной точках.

10. Последовательности из n единиц и n минус единиц, все частичные суммы которых неотрицательны.

Порядок проведения занятия

1. Построить схему рекурсивной композиции деревьев И/ИЛИ на основе рекуррентной формуле для чисел Каталана.
2. Построить автомат последовательной генерации.
3. Построить биекцию между деревом и комбинаторным множеством.
4. Построить алгоритмы комбинаторной генерации: нумерации, генерации по номеру, последовательной генерации.
5. Реализовать функции, отладить.
6. Провести исследование полученных алгоритмов.

2.14 Практическое занятие «Алгоритмы комбинаторной генерации корневых деревьев»

Теоретические основы

Рассмотрим применение деревьев И/ИЛИ для генерации деревьев с различными свойствами: двоичных деревьев с высотой не более

n+1, АВЛ-деревьев, двоичных деревьев с *n* узлами, двоичных деревьев высотой *h*, 2-3 деревьев и др. Структуры и алгоритмы для представления деревьев достаточно полно описаны в работе А. Ахо и др.

В связи с этим, добавим некоторые новые элементы в язык записи алгоритмов:

Node задает описание узла дерева. Для двоичных деревьев

Node.left – задает связь с левым сыном;

Node.right –задает связь с правым сыном;

Для узлов, у которых число связей больше 2, используется обозначение:

Node[1] – задает связь на самого левого сына;

Node[*i*] – задает связь на *i*-го сына (нумерация слева направо).

Отсутствие связи обозначается значением *Nil*.

Функция *CreateNode* – создает узел дерева, при этом все связи равны *Nil*.

Генерация и нумерация двоичных деревьев. Двоичным деревом называется дерево, у которого узел содержит не более двух сыновей.

Генерация двоичных деревьев высотой не более чем n+1

Рассмотрим правила *построения дерева И/ИЛИ* для генерации двоичных деревьев высотой не более *n*+1. Для этого запишем следующие случаи для узла двоичного дерева:

1. Узел является листом (Р1).

2. Узел имеет левого сына (Р2).

3. Узел имеет правого сына (Р3).

4. Узел имеет двух сыновей (Р4).

Дерево И/ИЛИ строится по следующим правилам:

1.Корень $\{\omega(n+1)\}$ является ИЛИ-узлом и имеет четыре сына, соответствующие случаям 1-4.

2. Сын, соответствующий случаю 1, имеет один лист.

3. Сыновья, соответствующие случаям 2 и 3, имеют одного сына, к которому можно применить описанные выше случаи.

4. Сын, соответствующий 4 случаю, имеет два сына, к которым также применяются, описанные выше четыре случая, и является И-узлом.

Применяя правила P1-P4, ко всем сыновьям соответствующих узлов P1-P4, получим схему рекурсивной композиции для построения дерева И/ИЛИ, показанное на рисунке 2.7.

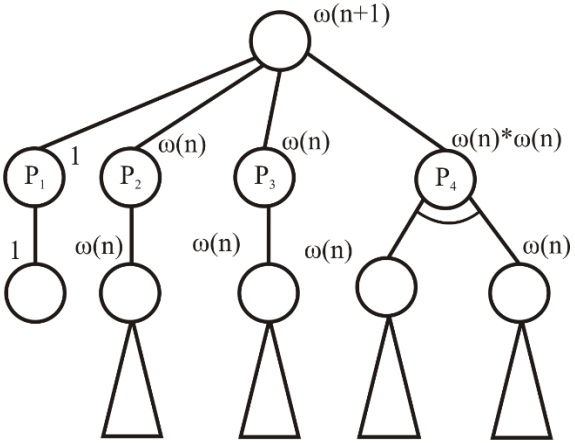

Рис.2.7 – Схема рекурсивной композиции дерева И/ИЛИ для построения алгоритма генерации двоичных деревьев

Получим теперь формулу подсчета вариантов.

Используя алгоритм подсчета вариантов в дереве И/ИЛИ, получим

$\omega(n+1) = \omega(p_1) + \omega(p_2) + \omega(p_3) + \omega(p_4)$, далее

$\omega(p_1) = 1$,

$\omega(p_2) = \omega(n)$,

$\omega(p_3) = \omega(n)$,

$\omega(p_4) = \omega(n) \cdot \omega(n)$.

Таким образом

$$\omega(n+1) = 1 + 2 \cdot \omega(n) + \omega(n) \cdot \omega(n) = [1 + \omega(n)]^2.$$

Построение алгоритма генерации

algorithm GenBinaryTree(*num*,*n*)

begin

if n=0 **then return** Nil;

Create(node); { создать узел дерева }

if *num*<1 **then return** Nil; { сыновей нет }

if $num < \omega(n-1)+1$ **then** { есть левый сын}

begin

node.left:=GenBinaryTree(*num*-1,n-1);

end

else

if $num < 2 \cdot \omega(n-1)+1$ **then** { есть только правый сын }

begin

node.right:=GenBinaryTree($num - \omega(n-1) - 1$,*n*-1);

end

else { есть два сына }

begin

$num := num - 2 \cdot \omega(n-1) - 1$

$l_1 := (num) \bmod \omega(n-1)$

$l_2 := (num) / \omega(n-1)$

node.left:=GenBinaryTree(l_1,*n*-1);

node.right:=GenBinaryTree(l_2,*n*-1);

end

return node;

end

Алгоритм нумерации:

1. По двоичному дереву строится вариант:

- если рассматриваемый узел – лист, то применяется правило Р1;
- если у рассматриваемого узла имеется только левый сын, то – правило Р2;
- если у рассматриваемого узла имеется только правый сын, то – правило Р3;
- если есть два сына, то правило Р4;
- далее рекурсивно спускаемся на уровень n-1.

2. Для определения номера необходимо знать все номера в варианте, поэтому производится рекурсивный спуск до листьев, затем вычисляются значения номеров для листьев (все номера для листьев равны 0), затем пересчитываем в номера соответствующих узлов-предков.

```
algorithm RankBinaryTree(Node,n)
 begin
  if Node=Nil Or n=0 then return 0;
  if Node.right=Nil and Node.left=Nil then return 0;
  if Node.right=Nil and Node.left<>Nil  then
    return RankBinaryTree(Node.left,n-1)+1;
  if Node.right<>Nil and Node.left=Nil  then
    return RankBinaryTree(Node.left,n-1)+ ω(n − 1) +1;
  else begin
    l1:=RankBinaryTree(Node.left,n-1)+1;
        l2:=RankBinaryTree(Node.left,n-1)+1;
    return l1+ ω(n − 1) *l2+2 ω(n − 1) +1;
  end
end
```

Ниже на рисунке 2.8 представлен результат работы алгоритма генерации для n=3.

Генерация двоичных деревьев с заданным числом узлов

Для корня двоичного дерева с n+1 узлами рассматриваются следующие варианты:

1) у корня имеется только правый сын, у которого n узлов;

2) у корня имеется два сына, у левого сына – 1 узел, у правого сына– n-1 узлов;

3) у корня имеется два сына, у левого сына – i узлов, у правого –n-i узлов.

4) у корня имеется только левый сын, у которого n узлов;

Всего правил будет $\{P_i\}_{i=0}^{n}$.

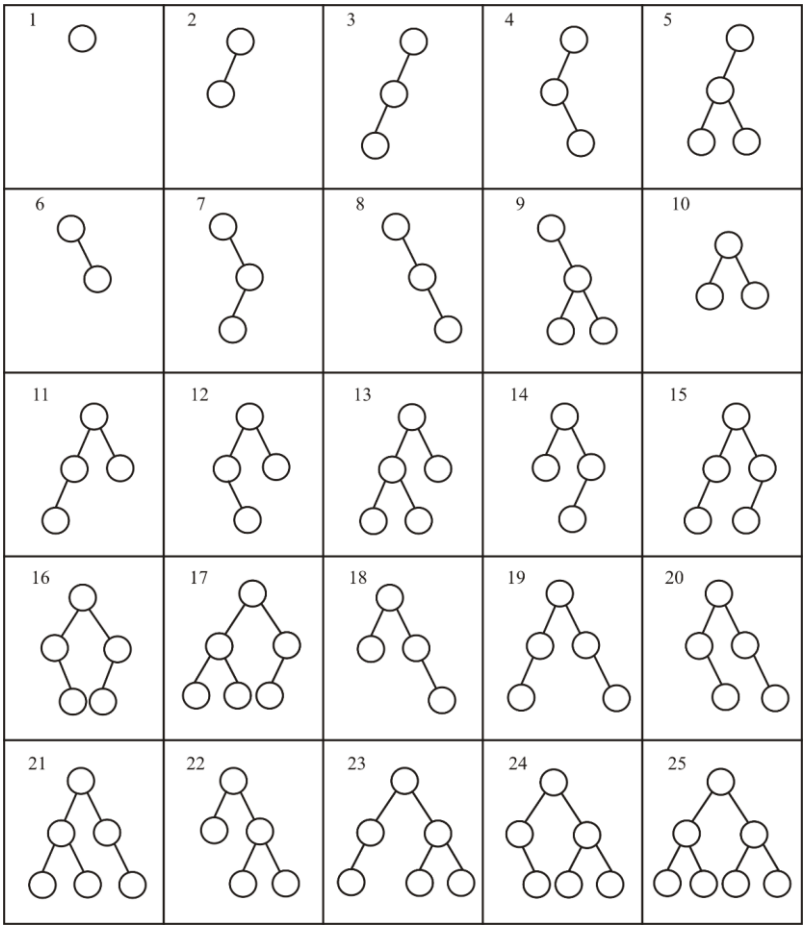

Рис.2.8 – Результат работы алгоритма генерации двоичных деревьев для случая n=3

Тогда для построения И/ИЛИ дерева запишем следующие правила:

1. Узел дерева {n+1} является ИЛИ узлом, у которого будет {n+1} сыновей соответствующим правилам $\{P_i\}_{i=0}^{n}$.

2. Узлы, соответствующие правилам, являются И-узлами и имеют двух сыновей, каждый из которых описывает соответственно левое или правое поддерево узла {n+1}.

Таким образом, рекурсивно применяя соответствующие правила $\{P_i\}_{i=0}^{n}$ получим схему рекурсивной суперкомпозиции построения дерева И/ИЛИ, изображенную на рисунке 2.9.

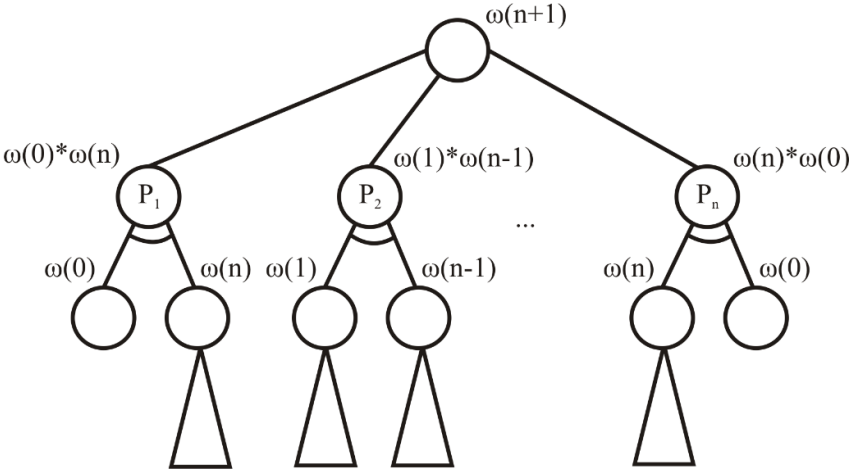

Рис.2.9 – Схема рекурсивной композиции построения дерева И/ИЛИ для генерации двоичных деревьев с n+1 узлами

Применяя алгоритм подсчета вариантов для дерева И/ИЛИ, получим следующее рекуррентное уравнение:

$$\omega(n+1) = \omega(0)\cdot\omega(n) + \omega(1)\cdot\omega(n-1) + \ldots + \omega(n-1)\cdot\omega(1) + \omega(n)\cdot\omega(0)$$

или

$$\omega(n+1) = \sum_{i=0}^{n}\omega(i)\cdot\omega(n-i),$$

$$\omega(0) = 1.$$

Эта формула идентична рекуррентному соотношению для последовательности чисел Каталана. Соответственно,

$$\omega(n) = \frac{(2n)!}{n!(n+1)!}.$$

Общее число листьев в дереве И/ИЛИ будет равно

$$L(n+1) = 2\sum_{i=0}^{n}L(i),$$

$$L(0) = 1.$$

Построение алгоритма генерации двоичного дерева по номеру num и с числом узлов равным n.

algorithm GenCatalan(num,n)

begin

if n=0 **then return** Nil;

Create(node)

S:=0;

for(i:=0,n-1) **begin**

$S := S + \omega(i) \cdot \omega(n - i - 1)$

if num<S **then** {определяем ИЛИ-узел}

begin

$l_1 := (num) \bmod \omega(i)$

$l_2 := (num) / \omega(i)$

node.left:=GenCatalan(l_1 ,i);

node.right:=GenCatalan(l_2 ,*n*-i-1);

return;

end

end

end

Рассмотрим теперь алгоритм нумерации. По конкретному двоичному дереву легко построить вариант. Однако для каждого узла двоичного дерева требуется знать число узлов, содержащихся в его левом и правом поддеревьях. Поэтому при рекурсивном спуске функция *Rank* возвращает два числа: номер соответствующего сына и число узлов в поддереве. Тогда алгоритм нумерации в соответствии со схемой рекурсивной суперкомпозиции (рис. 2.9) будет следующий:

algorithm RankCatalan(node,n)

begin

if n=0 or node=Nil **then return** (0,1); { используем правило для листа }

(l1,k):=RankCatalan(node.left,n-1); { вычисляем пару для левого сына }

(l2,m):=RankCatalan(node.right,n-1); { вычисляем пару для правого сына }

l:=l1+l2$\omega(k)$; { вычисляем номер для И-узла }

S:=0;

if k>0 **then**

for(i:=0,k) **begin** { вычисляем номер для узла ИЛИ }

$S := S + \omega(i) \cdot \omega(n-k-1)$

end

return (l+S,k+m+1);

end

Ниже на рисунке 2.10 представлен результат работы алгоритмов генерации и нумерации для *n*=4.

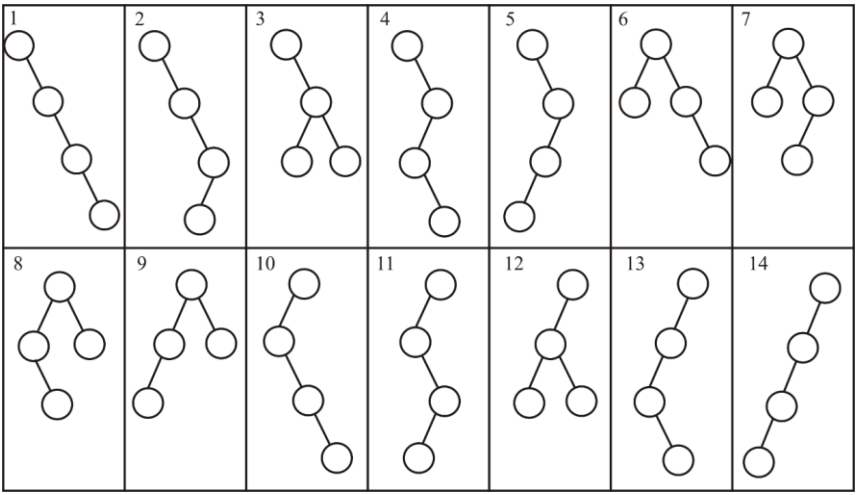

Рис.2.10 – Результат работы алгоритма для n=4

Генерация АВЛ-деревьев с заданной высотой n

АВЛ-дерево – это двоичное дерево, у которого высота поддеревьев левого и правого сына отличаются на единицу. Данный тип двоичного дерева позволяет эффективно организовать поиск информации.

Построение дерева И/ИЛИ

Для каждого узла АВЛ-дерева рассматривается три случая:

1) высота поддерева левого сына меньше высоты поддерева правого сына;

2) высота поддерева правого сына меньше высоты поддерева левого сына;

3) высоты поддеревьев сыновей равны.

Тогда И/ИЛИ дерево строится по следующим правилам:

1. Корень дерева является ИЛИ узлом и содержит три сына {P1},{P2},{P3}, каждый из которых соответствует случаю 1-3.

2. Каждый узел {P1},{P2},{P3} является И-узлом и имеет два сына, которые соответствуют левому и правому поддереву АВЛ-дерева.

3. Применяя рекурсивно эти правила, получим схему рекурсивной суперкомпозиции дерева И/ИЛИ, (рис 2.11.)

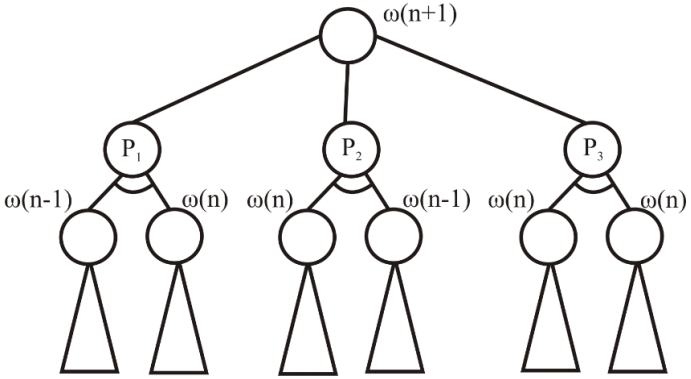

Рис.2.11 – Схема рекурсивной суперкомпозиции для дерева И/ИЛИ генерации АВЛ-деревьев высотой *n*+1

Теперь, используя метод подсчета вариантов для схемы рекурсивной композиции, запишем выражение для подсчета числа вариантов для СРС, представленной на рис 10.

$\omega(n+1) = \omega(p_1) + \omega(p_2) + \omega(p_3)$, где

$\omega(p_1) = \omega(n-1)\omega(n)$,

$\omega(p_2) = \omega(n)\omega(n-1)$,

$\omega(p_3) = \omega(n)\omega(n)$,

суммируя, получим следующее рекуррентное соотношение для подсчета АВЛ-деревьев высотой *n*+1:

$$\omega(n+1)=\begin{cases}\omega(0)=1,\\ \omega(1)=1,\\ \omega^2(n)+2\omega(n)\omega(n-1).\end{cases}$$

Эта формула совпадает с формулой, полученной Д.Кнутом.

Алгоритм генерации АВЛ-деревьев

algorithm AVLGenerate(*num*, *n*) **begin**

if n==0 **then return Nil**;

Create(Node)

if(h==1) **then return** Node;

if $num < \omega(n-2)\omega(n-1)$ **then** { правило p1 }

begin

$l_1 := (num) \bmod \omega(n-2)$

$l_2 := (num) / \omega(n-2)$

node.left:= AVLGenerate(l_1 ,n-2);

node.right:=AVLGenerate(l_2 ,*n*-1);

end

else

if $num < 2\omega(n-2)\omega(n-1)$ **then** { правило p2 }

begin

$num := num - \omega(n-2)\omega(n-1)$

$l_1 := (num) \bmod \omega(n-1)$

$l_2 := (num) / \omega(n-1)$

node.left:= AVLGenerate(l_1 ,n-1);

node.right:=AVLGenerate(l_2 ,*n*-2);

end

else begin { правило p3 }

$num := num - 2\omega(n-2)\omega(n-1)$

$num := num - \omega(n-2)\omega(n-1)$

$l_1 := (num) \bmod \omega(n-1)$

$l_2 := (num) / \omega(n-1)$

node.left:= AVLGenerate(l_1,n-1);

node.right:=AVLGenerate(l_2,*n*-1);

end

return Node

end

Алгоритм нумерации АВЛ-дерева высотой *n* строится в соответствии с методом построения алгоритмов нумерации. В основе алгоритма лежит схема рекурсивной суперкомпозиции дерева И/ИЛИ для АВЛ-деревьев (рис. 2.11). Используя рекурсивный спуск, можно построить вариант и найти сопоставление в дереве И/ИЛИ, для этого функция *Rank* вычисляет номер узла и высоту АВЛ-поддерева этого узла. Далее, имея полученные значения номеров для левого и правого поддеревьев и соответствующие высоты, определяется номер и высота рассматриваемого узла.

Запишем этот алгоритм:

Основные обозначения: l_{left} – номер левого сына; h_{left} – высота поддерева левого сына; l_{right} – номер правого сына; h_{right} – высота поддерева правого сына.

algorithm RankAvlTree(Node,n) **begin**

if n=0 Or Node=Nil **return** (0,0);

(l_{left}, h_{left}):=RankAvlTree(Node.left,n-1)

(l_{right}, h_{right}):=RankAvlTree(Node.right,n-1)

if $h_{left} < h_{right}$ **then** { правило 1 }

return $l_{left} + \omega(n-2) \cdot l_{right}, h_{right} + 1$;

if $h_{left} > h_{right}$ **then** { правило 2 }

return $l_{left} + \omega(n-1) \cdot l_{right} + \omega(n-1) \cdot \omega(n-2), h_{left} + 1$;

{правило 3 }

return $l_{left} + \omega(n-1) \cdot l_{right} + 2\omega(n-1) \cdot \omega(n-2), h_{left} + 1;$

end

На рисунке 2.12 представлен результат работы генерации и нумерации для *n*=3

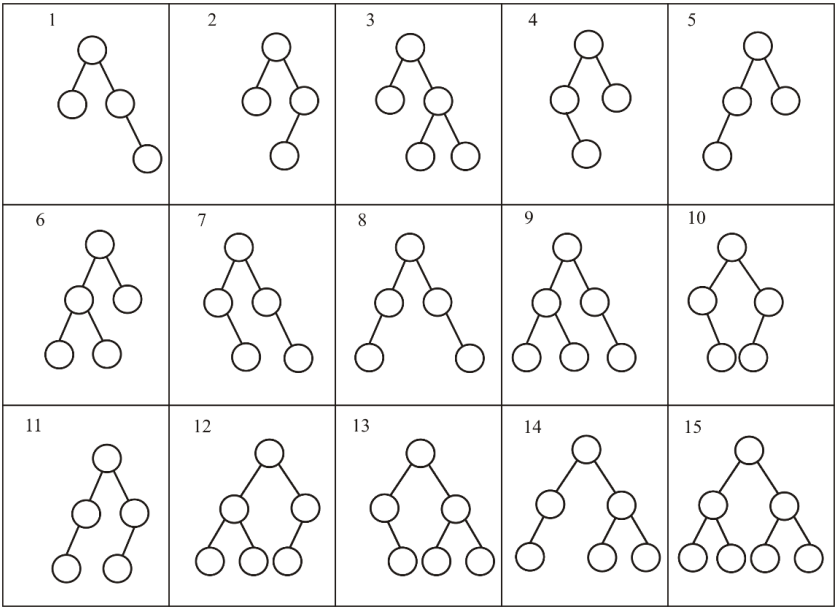

Рис. 2.12 – Результат работы алгоритма генерации для *n*=3

Генерация АВЛ деревьев с заданной высотой n и минимальным числом узлов

Построение дерева И/ИЛИ

Рассмотрим построение дерева И/ИЛИ для АВЛ-деревьев с минимальным числом узлов. Очевидно, что для построения такого дерева нужно из И/ИЛИ дерева (рис.2.11) исключить узлы для правила Р3 (это правило гласит, что высота левого и правого сына равна). Тогда получим следующую схему рекурсивной композиции И/ИЛИ дерева (рис. 2.13).

Тогда формула для подсчета вариантов будет

$$\omega(n) = 2\omega(n-1)\omega(n-2),$$

$\omega(0)=1$,

$\omega(1)=1$,

$\omega(2)=2$.

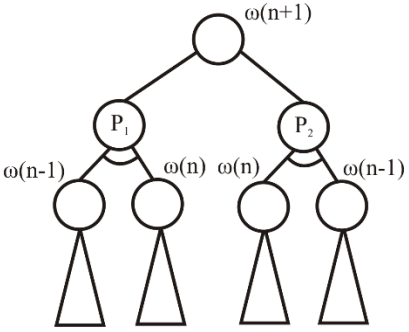

Рис. 2.13 – Схема рекурсивной композиции дерево И/ИЛИ для генерации АВЛ-деревьев с минимальным числом узлов и высотой *n*+1

Формула без рекурсии будет следующая.

$$\omega(n)=2^{F(n-2)+F(n-3)+1},\ n>2,$$

где $F(k)=F(k-1)+F(k-2)$ – числа Фибоначчи.

Тогда общее число листьев в дереве И/ИЛИ будет $L(n)=2(L(n-1)+L(n-2)+1)$.

Алгоритм генерации

Node AVLMinGenerate(*num*, *n*) **begin**

if n==0 **then return Nil**;

Create(Node)

if(n==1) **then return** Node;

if $num<\omega(n-2)\omega(n-1)$ **then** { правило p1 }

begin

$l_1 := (num) \bmod \omega(n-2)$

$l_2 := (num)/\omega(n-2)$

node.left:= AVLMinGenerate(l_1,*n*-2);

node.right:=AVLMinGenerate(l_2,*n*-1);

end

else

{ правило p2 }

begin

$num := num - \omega(n-2)\omega(n-1)$

$l_1 := (num) \bmod \omega(n-1)$

$l_2 := (num) / \omega(n-1)$

node.left:= AVLMinGenerate(l_1,*n*-1);

node.right:=AVLMinGenerate(l_2,*n*-2);

end

return Node

end

Следует отметить, что при *num*=0 мы получим дерево Фибоначчи для заданного параметра *n*.

Генерация двоичных деревьев высотой n

Построение дерева И/ИЛИ. Развивая идею построения И/ИЛИ дерева для генерации АВЛ-деревьев можно расширить условия, например, генерировать деревья, высоты поддеревьев левого и правого сына которых отличается на 2 на 3 и т.д. Тогда можно построить И/ИЛИ дерево для генерации двоичных деревьев высотой *n*+1. Для этого рассматриваются следующие случаи.

1) высота поддерева левого сына равна нулю, правого – *n*;

2) высота поддерева левого сына равна 1, правого– (*n*-1);

и т.д.

3) высота поддерева левого сына равна 2, правого –(*n*-2);

4) высота поддерева левого сына равна 3, правого – (*n*-3);

5) высота поддерева левого сына равна *i*, правого–(*n*-*i)*;

и т.д.

6) высота поддерева левого сына равна *n*, правого 0.

Применяя эти случаи для построения дерева И/ИЛИ, получим следующее схему рекурсивной композиции (рис. 2.14).

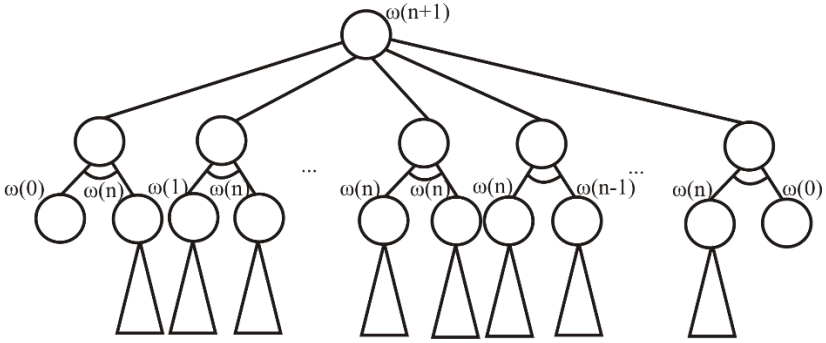

Рис. 2.14 – СРС дерева И/ИЛИ для генерации двоичного дерева высотой n+1.

Тогда общее число вариантов будет

$$\omega(n+1) = \omega^2(n) + 2\sum_{i=0}^{n-1}\omega(i)\omega(n), \quad \omega(0)=1, \quad \omega(1)=1 \qquad \textbf{(1).}$$

Известно и показано, что число деревьев высотой меньше или равно *n* определяется выражением

$$\omega_{bt}(n+1) = (\omega_{bt}(n)+1)^2, \; \omega_{bt}(0)=0 \qquad \textbf{(2).}$$

Следовательно, разность между $\omega_{bt}(n+1)$ и $\omega_{bt}(n)$ даст число двоичных деревьев высотой *n*+1

$$\omega(n+1) = \omega_{bt}(n+1) - \omega_{bt}(n).$$

Покажем, что выражения (1) и (2) тождественны. Для этого запишем

$$\omega(1) = \omega_{bt}(1) - \omega_{bt}(0),$$

$$\omega(2) = \omega_{bt}(2) - \omega_{bt}(1),$$

$$\omega(3) = \omega_{bt}(3) - \omega_{bt}(2),$$

$$\ldots$$

$$\omega(n) = \omega_{bt}(n) - \omega_{bt}(n-1).$$

Просуммируем левые и правые части выражений и, приведя подобные члены, получим:

$$\sum_{i=1}^{n}\omega(i)=\omega_{bt}(n) \qquad (3)$$

Затем рассмотрим правую часть выражения (1) и произведем следующие действия:

$$\omega(n+1)=\omega^2(n)+2\sum_{i=0}^{n-1}\omega(i)\omega(n)+\left[\sum_{i=0}^{n-1}\omega(i)\right]^2-\left[\sum_{i=0}^{n-1}\omega(i)\right]^2,$$

$$\omega(n+1)=\left[\sum_{i=0}^{n}\omega(i)\right]^2-\left[\sum_{i=0}^{n-1}\omega(i)\right]^2,$$

$$\omega(n+1)=\left[\sum_{i=1}^{n}\omega(i)+\omega(0)\right]^2-\left[\sum_{i=1}^{n-1}\omega(i)+\omega(0)\right]^2,$$

Используя выражение (3), произведем замену

$$\omega(n+1)=[\omega_{bt}(n)+1]^2-[\omega_{bt}(n-1)+1]^2$$

Затем, используя выражение для числа деревьев высотой меньше или равно *n*, получим, что

$$\omega(n)=\omega_{bt}(n)-\omega_{bt}(n-1)\cdot$$

Этот пример показывает, что, если множество задано как разность двух множеств, представленных деревьями И/ИЛИ, то и разность можно представить в виде дерева И/ИЛИ.

Построение алгоритма генерации

algorithm GenBinTreeH(*num*,*n*) **begin**

if *n*=0 then return Nil;

Create(Node)

Sum:=0

for(*k*:=0,*n*-1) **begin**

if num<Sum+ $\omega(k)\omega(n-1)$ **then** { определяем ИЛИ-узел}

begin

num:=num-Sum

$l_1 := (num) \bmod \omega(k)$

$l_2 := (num)/\omega(k)$

node.left:= GenBinTreeH(l_1,*k*);

node.right:=GenBinTreeH(l_2 ,*n*-1);

return Node;

end

num:=num+ $\omega(k)\omega(n-1)$;

end

for(*k*:=*n*-2,0) **begin**

if num< Sum+ $\omega(n-1)\omega(k)$ **then** { определяем ИЛИ-узел}

begin

num:=num-Sum

$l_1 := (num) \bmod \omega(n-1)$

$l_2 := (num) / \omega(n-1)$

node.left:= GenBinTreeH(l_1 **,n-1**);

node.right:=GenBinTreeH(l_2 **,k**);

return Node;

end

num:=num+ $\omega(n-1)\omega(k)$;

end

end

Генерация полных двоичных деревьев высотой не более n+1

Определение полного двоичного дерева было дано ранее

Построение дерева И/ИЛИ. Для построения дерева И/ИЛИ используется всего два правила:

1. Узел не имеет сыновей, это лист.

2. Узел имеет ровно два сына.

Схема рекурсивной композиции дерева И/ИЛИ для такого случая представлено на рисунке 2.15.

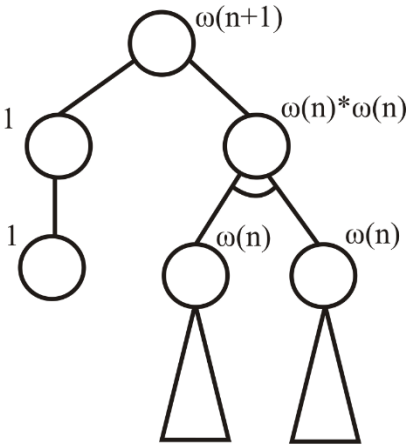

Рис. 2.15 – Схема рекурсивной композиции дерева И/ИЛИ для генерации полных двоичных деревьев высотой не более n+1

Число вариантов в дереве будет равно

$$\omega(n+1)=1+\omega^2(n),\ \omega(1)=1.$$

Алгоритм генерации

Это частный случай двоичного дерева, поэтому можно воспользоваться алгоритмом GenBinaryTree(*num*,*n*). Для этого из алгоритма удаляются ветви алгоритма для случаев, когда у узла один или левый или правый сын. Тогда получим следующий алгоритм:

algorithm GenFullBinaryTree(*num*,*n*)

begin

if n=0 **then return** Nil;

Create(node); { создать узел дерева }

if *num*<1 **then return** Nil; { сыновей нет }

begin { есть два сына }

$num := num-1$

$l_1 := (num) \bmod \omega(n-1)$

$l_2 := (num)/\omega(n-1)$

node.left:=GenFullBinaryTree(l_1,*n*-1);

node.right:=GenFulBinaryTree(l_2,*n*-1);

```
    end
    return node;
  end
```

Аналогично получается алгоритм нумерации полных двоичных деревьев.

Порядок проведения занятия

1. Построить схему рекурсивной композиции деревьев И/ИЛИ на основе анализа правил построения дерева.
2. Построить автомат последовательной генерации.
3. Построить биекцию между деревом и комбинаторным множеством.
4. Построить алгоритмы комбинаторной генерации: нумерации, генерации по номеру, последовательной генерации.
5. Реализовать функции, отладить.
6. Провести исследование полученных алгоритмов.

2.15 Практическое занятие «Алгоритмы, основанные на генетическом методе»

Порядок проведения занятия

1. Изучить генетический метод построения алгоритмов оптимизации.
2. Разработать алгоритм на генетическом методе.
3. Разработать и отладить программу.
4. Провести исследования алгоритма.

Рекомендуемые источники

1. Мещеряков, П. С. Прикладная информатика: Методические указания по курсовому проекту [Электронный ресурс] / П. С. Мещеряков, С. В. Тимченко – Томск: ТУСУР, 2012. – 30 с. – Режим доступа: https://edu.tusur.ru/publications/1766 (дата обращения: 15.05.2018).

2.16 Практическое занятие «Комбинированные методы построения алгоритмов»

Порядок проведения занятия

1. Изучить примеры комбинированных алгоритмов:
   - Комбинированный рекурсивно-итерационный алгоритм сортировки.
   - Комбинированный алгоритм решения классической задачи одномерной упаковки.

- Рациональные ресурсно-адаптивные алгоритмические решения по компоненту формирования глобальной матрицы в системе конечно элементного анализа.
- Рациональная организация аналитической базы данных с использованием алгоритмов решения задачи упаковки с динамической внутренней границей объема.

2. Выбрать, разработать и отладить программу.
3. Построить схему вычислительного эксперимента.
4. Представить алгоритм, программу и график вычислительной сложности в отчет.

Рекомендуемые источники

1. Ульянов, М.В. Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ [Электронный ресурс] : учеб. пособие – Электрон. дан. – М. : Физматлит, 2008. – 304 с. – Режим доступа: https://e.lanbook.com/book/2354. (дата обращения: 15.05.2018).

2.17 Практическое занятие «Метод тестирования белым ящиком»

Теоретические основы

Метод белого ящика (white box testing, open box testing, clear box testing, glass box testing) – метод, когда у тестировщика есть доступ к внутренней структуре и коду приложения, а также есть достаточно знаний для понимания увиденного.

Разработка тестов методом белого ящика (white-box test design technique) **–** процедура разработки или выбора тестовых сценариев на основании анализа внутренней структуры компонента или системы.

Тестирование методом белого ящика, основывается на конкретной структуре программного продукта или системы:

- Компонентный уровень: структура компонента программного обеспечения, т.е. операторы, альтернативы, ветви или определенные пути.
- Интеграционный уровень: структура может быть представлена деревом вызовов (диаграмма, в которой модули вызывают другие модули).
- Системный уровень: структура может представлять собой структуру меню, бизнес-процессов или же схему веб-страницы.

Техники, основанные на структуре, или методе белого ящика:

- покрытие операторов;

- покрытие альтернатив;
- покрытие решений.

Покрытие кода (code coverage) – метод анализа, определяющий, какие части программного обеспечения были проверены (покрыты) набором тестов, а какие нет, например, покрытие операторов, покрытие альтернатив или покрытие условий.

Покрытие операторов (statement coverage) – процентное отношение операторов, исполняемых набором тестов, к их общему количеству.

При тестировании операторов тестовые сценарии создаются таким образом, чтобы выполнять определенные операторы и обычно увеличивать покрытие операторов. Величина покрытия операторов определяется как отношение числа выполняемых операторов, покрытых тестовыми сценариями (разработанными или выполненными) к общему числу операторов в тестируемом коде.

Покрытие альтернатив (decision coverage) – процент результатов альтернативы, который был проверен набором тестов. Стопроцентное покрытие решений подразумевает стопроцентное покрытие ветвей и стопроцентное покрытие операторов.

В методе тестирования альтернатив тестовые сценарии создаются для выполнения определенных результатов альтернатив. Ветви исходят из точек альтернатив в программном коде и показывают передачу управления различным участкам кода. Покрытие альтернатив определяется отношением числа всех результатов альтернатив, покрытых разработанными или выполненными тестовыми сценариями к числу всех возможных результатов альтернатив в тестируемом коде.

Модульное тестирование, или *юнит-тестировние* (*unit testing*) – процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

Идея состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к *регрессии*, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

Цель *модульного тестирования* – изолировать отдельные части программы и показать, что по отдельности эти части работоспособны.

Существует различные инструменты как *JUnit, PHPUnit, TestNG, PyTest*, которые позволяют создавать и поддерживать качественные юнит-тесты.

Рекомендуется использовать следующие инструменты:

- JUnit.
- TestN.
- Code coverage.

JUnit – это Java фреймворк для тестирования, т. е. тестирования отдельных участков кода, например, методов или классов. Опыт, полученный при работе с JUnit, важен в разработке концепций тестирования программного обеспечения.

Junit предоставляет нам следующие возможности:

- Базовые классы и Аннотации для написания юнит-тестов.
- Базовый класс для запуска тестов – TestRunner class.
- Поддержка классов и аннотаций для написания тест-сьютов – @RunWith(Suite.Class).
- Отчет результатов тестирования.

JUnit Annotations – процесс добавления специальных синтаксических форм метаданных в Java. JUnit Annotations: переменные, параметры, пакеты, методы и классы.

Проверки чаще всего выполняются с помощью класса **Assert** хотя иногда используют ключевое слово assert.

Чтобы ускорить процесс и сделать его более автоматизированным используют параметризированные тесты (@Parameterized.Parameters). С их помощью можно создать тестовой класс, и тестировать модуль, используя различные данные с помощью тестового класса

С помощью @RunWith можно аннотировать тестовый класс, передав этой аннотации параметром значение *Parameterized.class*.

```
   @RunWith(value = Parameterized.class)
   public class ParametersTest {
// ...
}
```

TestNG – это фреймворк для тестирования, написанный Java, он взял много чего с JUnit и NUnit, но имеет более гибкий и расширенный функционал.

Принцип работы с TestNG очень схож с JUnit.

Написать автотесты – это еще полдела, необходимо проверить, а весь ли код покрыт тестами. Автоматические тесты должны покрывать 100% функционала. Данная характеристика называется «*code coverage*» и буквально означает степень покрытия кода тестами.

Различаются следующие показатели:

- Function Coverage – подсчёт по вызовам методов
- Decision Coverage – подсчёт по возможным направлениям исполнения кода (then-else или case-case-default в управляющих структурах). Учитывает единственное ветвление в каждом конкретном случае.
- Statement Coverage – подсчёт по конкретным строчкам кода
- Path Coverage – подсчёт по возможным путям исполнения кода. Более широкое понятие, чем decision coverage, так как учитывает результат всех ветвлений.
- Conditional Coverage – подсчёт по возможным результатам вычисления значениям булевских выражений и подвыражений в коде.

Покрытие кода (инструменты):

- Java (Jcov, *EclEmma – Java Code Coverage for Eclipse*).
- C++ (Gcov, Tcov).
- C# (OpenCover).
- встроенный в Visual Studio Test Coverage.

EclEmma – бесплатный инструмент покрытия кода на Java для среды *Eclipse*, доступный по лицензии Eclipse Public License.

Достоинство *EclEmma*:

- Доступный анализ покрытия кода прямо в IDE Eclipse.
- Запуски тестов типа JUnit в Eclipse могут быть проанализированы напрямую на предмет наличия покрытия кода.
- Не требует модификации проектов или выполнения всяких установок.
- Результат оценки покрытия виден в редакторе кода, при этом настраиваемый цвет кода показывает полностью покрытые строки кода, частично покрытые и не покрытые тестами строки кода.
- Позволяет экспортировать **отчёты о покрытии:** данные о покрытии могут быть экспортированы в формате HTML, XML или CSV.

Следующие типы кода для запуска поддерживают EclEmma:

- Local Java application.
- Eclipse/RCP application.
- Equinox OSGi framework.
- JUnit test.

- TestNG test.
- JUnit plug-in test.
- JUnit RAP test.
- SWTBot test.
- Scala application.

Пример

Рассмотрим пример как можно протестировать код.

Шаг 1. Создадим проект.

Шаг 2. Напишем код или метод, который будем тестировать.

```
package test_meth;
public class method {
public static int alg(int A, int B, int X) {
      //точка a
      if((A>1)&&(B==0))
      X=X/A;//точка c
      //точка b
      if((A==2)||(X>1))
      X=X++; //точка e
      //точка d
      return X;
      } }
```

Шаг 3. Напишем тесты. Постараемся добиться полного покрытия кода. Добавим в проект **Junit**.

```
package test_meth;
import static org.junit.Assert.*;
import org.junit.Test;
public class methodTest {
      @Test
      public void test1() {
assertNotNull(null, new method().alg(2, 0, 4)); }
      @Test
      public void test2() {
assertNotNull(null, new method().alg(2, 1, 1)); }
      @Test
      public void test3() {
assertNotNull(null, new method().alg(1, 0, 2));}
      @Test
      public void test4() {
assertNotNull(null, new method().alg(1, 1, 1));}
```

```
	@Test
	public void test5() {
assertNotNull(null, new method().alg(0, 0, 0));
	} }
```

Можно запускать тесты вручную с помощью программного кода. Для этого можно воспользоваться **Runner**. Бывают текстовый – junit.textui.TestRunner, графические версии – junit.swingui.TestRunner, junit.awtui.TestRunner, или класс JUnitCore.

Добавим следующий метод в метод main() в класс methodTest, получим:

```
package test_meth;
import static org.junit.Assert.*;
import org.junit.Test;
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
public class methodTest {
	@Test
	public void test1() {
		new method();
assertNotNull(null, method.alg(2, 0, 4));
		 }
	@Test
	public void test2() {
		new method();
assertNotNull(null, method.alg(2, 1, 1));
	}
	@Test
	public void test3() {
		new method();
assertNotNull(null, method.alg(1, 0, 2));
	}
	@Test
	public void test4() {
		new method();
assertNotNull(null, method.alg(1, 1, 1));
	}
	@Test
	public void test5() {
	new method();
assertNotNull(null, method.alg(0, 0, 0));
	}
```

```
    public static void main(String[] args) throws
Exception {
       JUnitCore runner = new JUnitCore();
Result result=runner.run(methodTest.class);

System.out.println("run tests: " +
result.getRunCount());
System.out.println("failed tests: " +
result.getFailureCount());
System.out.println("ignored tests: " +
result.getIgnoreCount());
System.out.println("success: " +
result.wasSuccessful());
      } }
```

Результат выводится на консоль IDE (рис. 2.16).

Писать 5 тестов для одного метода слишком долго, а бывает их не 5, а гораздо больше. Лучше воспользоваться параметризированными тестами. Перепишем тесты:

```
package test_meth;
import static org.junit.Assert.*;
import org.junit.Test;
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.testng.Assert;
import java.util.Arrays;
@RunWith(Parameterized.class)
public class methodTest {
    private int valueA;
    private int valueB;
    private int valueX;
    private int expected;
    public methodTest(int valueA, int valueB, int
valueX, int expected) {
        this.valueA = valueA;
        this.valueB = valueB;
        this.valueX = valueX;
        this.expected = expected;
    }
@Parameterized.Parameters(name = "{index}: re-
zOf({0},{1},{2}=={3}")
```

```
public static Iterable<Object[]> dataForTest() {
        return Arrays.asList(new Object[][]{
                {2, 0, 4, 2},
                {2, 1, 1, 1},
                {1, 0, 2, 2},
                {1, 1, 1, 1},
                {0, 0, 0, 0}
        });
    }
      @Test
        public void test_peram() {
        new method();

Asert.assertEquals(expected,method.alg(valueA,valueB
, valueX));

      }

    public static void main(String[] args) throws
Exception {
 JUnitCore runner = new JUnitCore();
 Result result = runner.run(methodTest.class);

 System.out.println("run tests: " + re-
sult.getRunCount());
 System.out.println("failed tests: " + re-
sult.getFailureCount());
 System.out.println("ignored tests: " + re-
sult.getIgnoreCount());
 System.out.println("success: " + re-
sult.wasSuccessful());


      } }
```

Результат выполнения параметризированных тестов показан на рисунке 2.17.

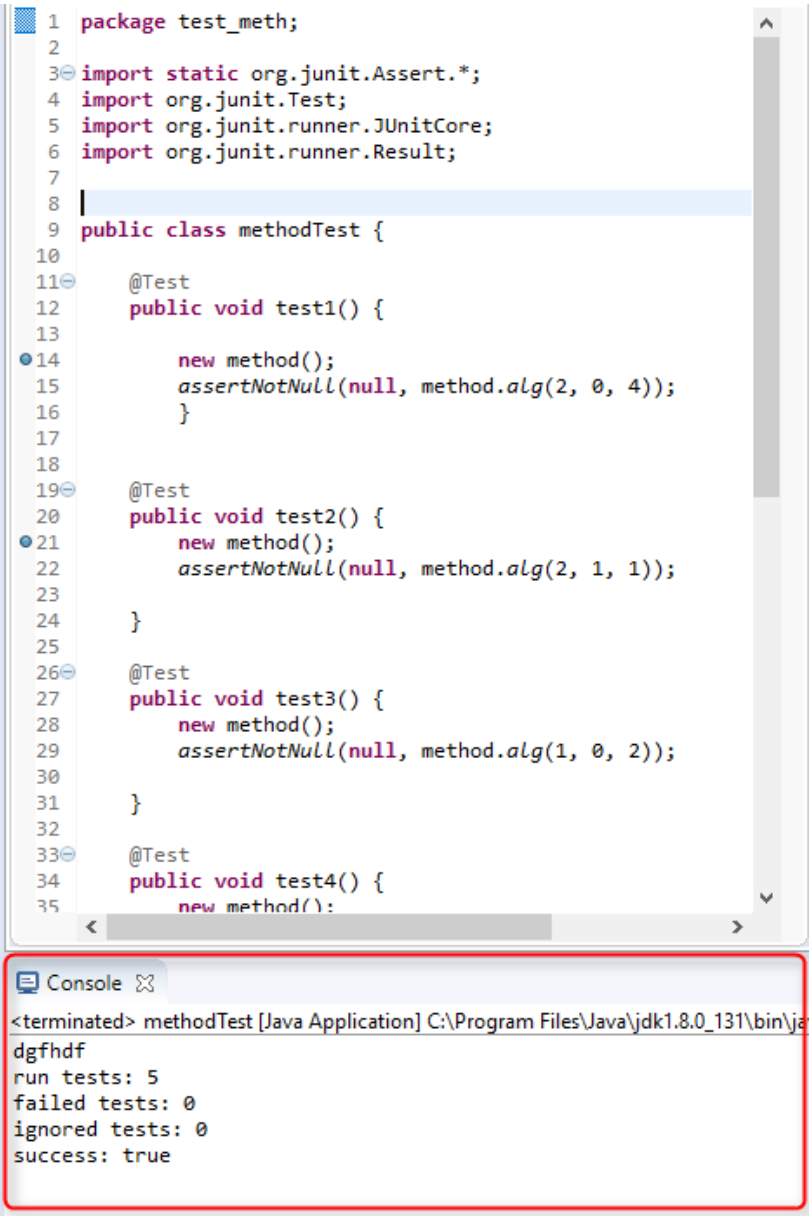
1 package test_meth;
2
3 import static org.junit.Assert.*;
4 import org.junit.Test;
5 import org.junit.runner.JUnitCore;
6 import org.junit.runner.Result;
7
8
9 public class methodTest {
10
11 @Test
12 public void test1() {
13
14 new method();
15 assertNotNull(null, method.alg(2, 0, 4));
16 }
17
18
19 @Test
20 public void test2() {
21 new method();
22 assertNotNull(null, method.alg(2, 1, 1));
23
24 }
25
26 @Test
27 public void test3() {
28 new method();
29 assertNotNull(null, method.alg(1, 0, 2));
30
31 }
32
33 @Test
34 public void test4() {
35 new method();
Console
<terminated> methodTest [Java Application] C:\Program Files\Java\jdk1.8.0_131\bin\ja
dgfhdf
run tests: 5
failed tests: 0
ignored tests: 0
success: true

Рис. 2.16

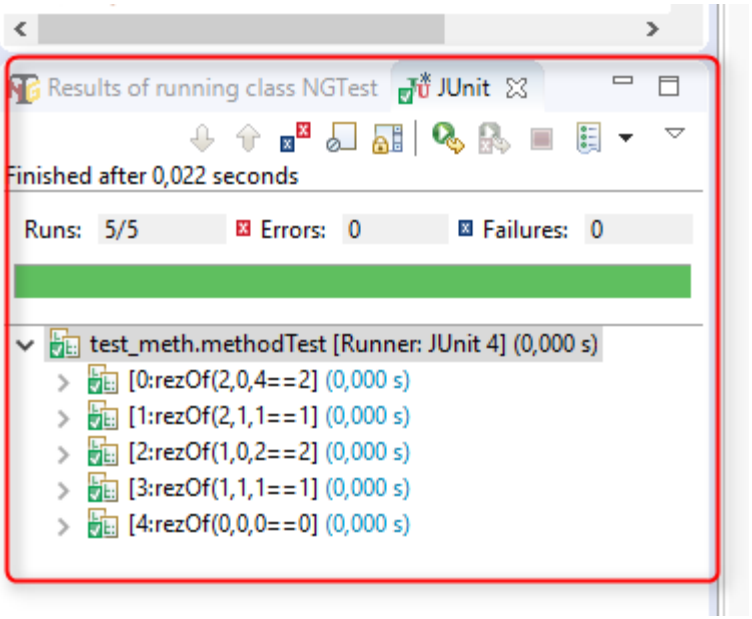

Рис. 2.17

Теперь давайте оценим покрытие кода (рис. 2.18).

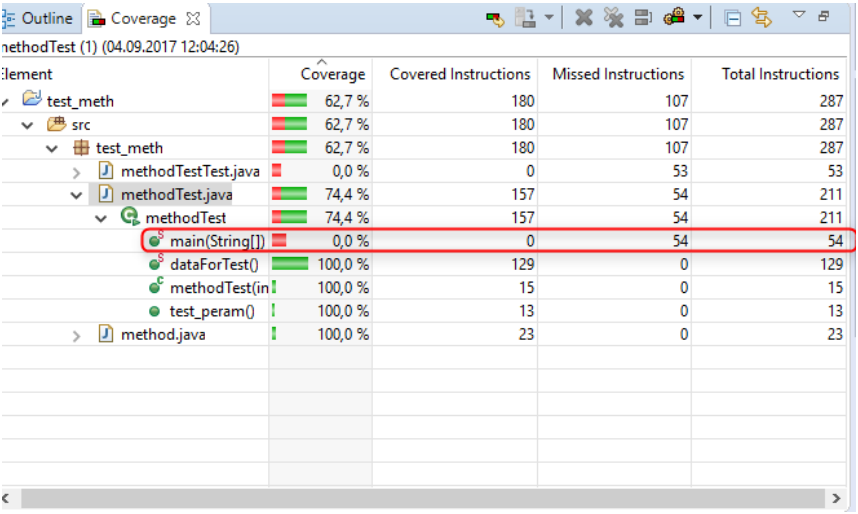

Рис. 2.18

Как видим, имеем покрытие нашего метода methodTest() 100%.

Варианты задания:

Вариант 1. Провести функциональное тестирование программы, которая решает квадратное уравнение.

Вариант 2. Провести функциональное тестирование программы, которая определяет вид треугольника, заданного длинами его сторон: равносторонний, равнобедренный, прямоугольный, разносторонний.

Вариант 3. Провести функциональное тестирование программы, которая из последовательности 10 целых чисел выводит максимальное значение элемента, минимальное значение элемента и их сумму.

Вариант 4. Провести функциональное тестирование программы, которая из последовательности 10 целых чисел выводит минимальное значение элемента, устанавливает, сколько раз это значение встречается в последовательности.

Вариант 5. Провести функциональное тестирование программы, которая из последовательности 10 целых чисел выводит значение элемента, повторяющееся большее число раз и выводит количество повторений в последовательности.

Вариант 6. Провести функциональное тестирование программы, которая из последовательности 10 целых чисел выводит разность между максимальным значением элемента и минимальным значением элемента.

Вариант 7. Провести функциональное тестирование программы, которая из последовательности 10 целых чисел выводит максимальное значение элемента, минимальное значение элемента и их произведение.

Вариант 8. Провести функциональное тестирование программы, которая из последовательности 10 целых чисел выводит минимальное значение элемента и проверяет, является ли это число простым.

Вариант 9. Провести функциональное тестирование программы, которая определяет вид четырехугольника, заданного координатами вершин на плоскости: квадрат, прямоугольник, параллелограмм, ромб, равнобедренная трапеция, прямоугольная трапеция, трапеция общего вида, четырехугольник общего вида.

Вариант 10. Провести функциональное тестирование программы, которая из последовательности 10 целых чисел выводит максимальное значение элемента и проверяет, является ли это число простым.

Рекомендованная литература

1. Котляров, В.П. Основы тестирования программного обеспечения [Электронный ресурс] : учеб. пособие. – М. : , 2016. – 248 с. – Режим доступа: https://e.lanbook.com/book/100352 (дата обращения: 24.04.2018).

2.18 Практическое занятие «Метод тестирования черным ящиком»

Теоретические основы

Чтобы облегчить жизнь тестировщику были разработаны различные техники и методы тест-дизайна, которые позволяют приблизится к исчерпывающему (полному) тестированию.

Метод чёрного ящика (black box testing, closed box testing, specification-based testing) – у тестировщика либо нет доступа к внутренней структуре и коду приложения, либо недостаточно знаний для их понимания, либо он сознательно не обращается к ним в процессе тестирования. Тестировщик не знает, как устроена тестируемая система.

Целью этой техники является поиск ошибок в таких категориях:

- неправильно реализованные или недостающие функции;
- ошибки интерфейса;
- ошибки в структурах данных или организации доступа к внешним базам данных;
- ошибки поведения или недостаточная производительности системы.

*Преимущества***:**

- тестирование производится с позиции конечного пользователя и может помочь обнаружить неточности и противоречия в спецификации;
- тестировщику нет необходимости знать языки программирования и углубляться в особенности реализации программы;
- тестирование может производиться специалистами, независимыми от отдела разработки, что помогает избежать предвзятого отношения;
- можно начинать писать тест-кейсы, как только готова спецификация.

Недостатки:

- тестируется только очень ограниченное количество путей выполнения программы;

• без четкой спецификации (а это скорее реальность на многих проектах) достаточно трудно составить эффективные тест-кейсы;

• некоторые тесты могут оказаться избыточными, если они уже были проведены разработчиком на уровне модульного тестирования;

Техники тест-дизайна, основанные на использования *черного ящика*:

• разбиение на классы эквивалентности;

• анализ граничных значений;

• попарное тестирование;

• таблицы решений.

Pairwise testing (*all-pairs analysis, попарное тестирование или попарный анализ, анализ всех пар комбинаций*) – это современная и эффективная методика тестирования, основанная на том предположении, что большинство дефектов возникает при взаимодействии не более двух факторов. Тестовые наборы, генерируемые при использовании данной методики, охватывают все уникальные пары комбинаций факторов, что считается достаточным для обнаружения большего числа дефектов.

Для попарного тестирования используются алгоритмы, основанные на построении ортогональных массивов или на *All-Pairs алгоритме*, которые опираются на теоретические исследования в области комбинаторных алгоритмов, алгоритмов дискретной математики.

All-Pairs algorithm (алгоритм всех пар) – это комбинаторная методика, которая была специально создана для попарного тестирования. В её основе лежит выбор возможных комбинаций значений всех переменных, в которых содержатся все возможные значения для каждой пары переменных.

Плюсы попарного тестирования следующие:

• Данный тип проверки уменьшает количество тест-кейсов необходимых для проверки продукта.

• Попарное тестирование ускоряет выполнение самого процесса контроля качества продукта.

• Практика показывает, что количество багов, обнаруженных с помощью попарного тестирования, будет больше, чем при проверке всех значений для каждого параметра ввода.

Есть множество инструментов для попарного тестирования. Есть и онлайновые сервисы: hexawise, inductive, testcover. Некоторые работают через консоль, другие через GUI. Главное – подать им на вход грамотный

набор данных. Удобно использовать консольную программу *Allpairs*, она не привязана ни к одной операционной системе.

Allpairs – программа, подбирающая уникальные пары для входящего набора данных. Работает из командной строки.

Порядок выполнения работы

1. Найти в своем проекте место, где может быть применим pairwise (много переменных, мало значений в каждой).

2. Составить входной файл, разбив входные данные на классы эквивалентности.

3. Выполнить программой *Allpairs* попарное тестирование.

4. Проанализировать результат.

5. Отправить отчет с входной и выходной таблицами с пояснениями.

Рекомендованная литература

Бейзер Б. Тестирование черного ящика. Технологии функционального тестирования программного обеспечения и систем. – СПб. : Питер, 2004. – 320 с.

3 Методические указания для организации самостоятельной работы

3.1 Общие положения

Целью самостоятельной работы является систематизация, расширение и закрепление теоретических знаний, использование материала, собранного и полученного в ходе самостоятельной подготовки к практическим занятиям.

Самостоятельная работа включает в себя подготовку к практическим занятиям, проработку лекционного материала и подготовку к контрольным работам, проработку тем дисциплины, вынесенных на самостоятельное изучение.

3.2 Проработка лекционного материала

Изучение теоретической части дисциплин призвано не только углубить и закрепить знания, полученные на аудиторных занятиях, но и способствовать развитию у студентов творческих навыков, инициативы и организовать свое время.

Проработка лекционного материала включает:

- чтение студентами рекомендованной литературы и усвоение теоретического материала дисциплины;
- знакомство с Интернет-источниками;
- подготовку ответов на вопросы по различным темам дисциплины в той последовательности, в какой они представлены.

Планирование времени, необходимого на изучение дисциплин, студентам лучше всего осуществлять весь семестр, предусматривая при этом регулярное повторение материала.

Материал, законспектированный на лекциях, необходимо регулярно прорабатывать и дополнять сведениями из других источников литературы, представленных не только в программе дисциплины, но и в периодических изданиях.

При изучении дисциплины сначала необходимо по каждой теме прочитать рекомендованную литературу и составить краткий конспект основных положений, терминов, сведений, требующих запоминания и являющихся основополагающими в этой теме для освоения последующих тем курса. Для расширения знания по дисциплине рекомендуется исполь-

зовать Интернет-ресурсы; проводить поиски в различных системах и использовать материалы сайтов, рекомендованных преподавателем.

3.3 Подготовка к практическим занятиям

Проведение практических занятий включает в себя следующие этапы:

- постановку темы занятий и определение задач практического занятия;
- определение порядка практического занятия или отдельных его этапов;
- непосредственное выполнение практического задания студентами и контроль за ходом занятий;
- подведение итогов практического занятия и формулирование основных выводов;
- оформление отчета и защиты практического задания (демонстрация работы и ответы на вопросы по теме занятия).

При подготовке к практическим занятиям необходимо заранее изучить методические рекомендации по его проведению. Обратить внимание на цель занятия, на основные вопросы для подготовки к занятию, на содержание темы занятия.

Если в процессе практического занятия или над изучением теоретического материала у студента возникают вопросы, разрешить которые самостоятельно не удается, необходимо обратиться к преподавателю для получения у него разъяснений или указаний.

3.4 Самостоятельное изучение тем теоретической части курса

Темы, отводимые на самостоятельное изучение:

1. Классификация тестов.

2. Методы и системы разработки компиляторов и интерпретаторов систем программирования.

Рекомендуемая литература

1. Проект Software-Testing.ru. – Режим доступа: http://software-testing.ru/ (дата обращения: 24.04.2018).

2. Стандартизация, сертификация и управление качеством программного обеспечения [Электронный ресурс] : учеб. пособие / Т.Н. Ананьева, Н.Г. Новикова, Г.Н. Исаев. – М.:НИЦ ИНФРА-М, 2016. - 232 с. – Режим

доступа: http://znanium.com/bookread2.php?book=541003 (дата обращения: 24.04.2018).

3. Рейуорд-Смит, В. Дж. Теория формальных языков. Вводный курс : Пер. с англ. / В. Дж. Рейуорд-Смит ; пер. Б. А. Кузьмин, ред. пер. Б. А. Шестаков. – М.: Радио и связь, 1988. – 124 с.

4. Льюис Ф., Розешкранц Д., Стирнз Р. Теоретические основы проектирования компиляторов. – М.: Мир, 1979. – 656 с.

5. Теория языков программирования и методы трансляции: Методическое пособие / Калайда В. Т. – 2012. 219 с. [Электронный ресурс] - Научно-образовательный портал ТУСУР – 2012. – Режим доступа: edu.tusur.ru/training/publications/2063 (дата обращения: 16.05.2018).

3.4.1 Классификация тестов

Классификация тестов позволяет упорядочить знания и значительно ускоряет процессы планирования тестирования и разработки тест-кейсов.

Для каждого уровня тестирования может быть определено: цели, артефакты процесса разработки, на основании которых будут разработаны тестовые сценарии, объекты тестирования, типичные дефекты и отказы, которые могут быть найдены во время тестирования.

Все виды тестирования можно условно разделить по *состоянию системы* на две большие группы:

1. статическое тестирование (*static testing*);

2. динамическое тестирование (*dynamic testing*).

Статическое тестирование – это процесс анализа самой разработки программного обеспечения, т. е. тестирование без запуска программы.

Динамическое тестирование – это тестовая деятельность, предусматривающая эксплуатацию (запуск) программного продукта. Динамическое тестирование предполагает запуск программы, выполнение всех её функциональных модулей и сравнение фактического её поведения с ожидаемым.

Тестирование ПО можно классифицировать по следующим признакам:

1. По знанию системы.

2. По позитивности.

3. По целям (объекту).

4. По исполнителям (субъекту).

5. По времени проведения.

6. По степени автоматизации.

Перечень вопросов, подлежащих изучению

1 Чем отличаются функциональное тестирование от нефункционального?

1. На какие уровни можно разделить функциональное тестирование?

2. В чем заключается разница между юзабилити тестированием и тестирование GUI?

3. Какие тесты лучшие кандидаты для автоматизации?

3.4.2 Методы и системы разработки компиляторов и интерпретаторов систем программирования

Разработка компиляторов и интерпретаторов является необходимым элементом современных программных систем, обеспечивающие эффективное использование временных, информационных, вычислительных ресурсов.

Разработка такого программного обеспечения носит сложный итерационных характер. Основные этапы разработки:

1. Анализ предметной области автоматизации.
2. Построение грамматики.
3. Выбор схемы реализации.
4. Построение отдельных блоков схемы реализации.
5. Комплексная отладка и тестирование.

Перечень вопросов, подлежащих изучению

1. Каким образом строить и анализировать грамматику проблемно-ориентированного языка

2. Как выбрать или разработать общей схемы для интерпретатора?

3. Как выбрать или разработать общей схемы для компилятора?

4. Как отладить и оттестировать соответствующее программное обеспечение?

4 Рекомендуемые источники

1. Методы построения алгоритмов генерации и нумерации комбинаторных объектов на основе деревьев И/ИЛИ : / В. В. Кручинин ; Томский государственный университет систем управления и радиоэлектроники (Томск). – Томск : В-Спектр, 2007. – 199[1] с.
2. Кнут, Д. Искусство программирования для ЭВМ. Т. 1. Основные алгоритмы / Д. Кнут. – М.: Мир, 1976. – 736 с.
3. Башмаков, М.И. Информационная среда обучения / М.И. Башмаков, С.Н. Поздняков, Н.А. Резник. – СПб.: СВЕТ, 1997. – 400 с.
4. Гигиенические требования к видеодисплейным терминалам, персональным электронно-вычислительным машинам и организации работы : Санпин 2.2.2.542-96 : утв. Постановлением Госкомсанэпиднадзора РФ от 14.07.96. п. 14.
5. Силич М.П., Силич В.А., Основы теории систем и системного анализа: учеб. пособие. – Томск: Изд-во Томск. гос. ун-та систем управления и радиоэлектроники, 2013. – 340 с.
6. Кручинин В.В. Разработка компьютерных учебных программ – Томск:Изд-во ТГУ, 1998г. – 212 с.
7. Гарайс, Д. В. Новые технологии в программировании: Учебное пособие [Электронный ресурс] / Д. В. Гарайс, А. Е. Горяинов, А. А. Калентьев – Томск: ТУСУР, 2014. – 176 с. – Режим доступа: https://edu.tusur.ru/publications/5796 (дата обращения: 15.05.2018).
8. Салмина Н. Ю., Функциональное программирование и интеллектуальные системы: учебное пособие [Электронный ресурс] / Салмина Н. Ю. – Томск: ТУСУР, 2016 . – 100 с. – Режим доступа: https://edu.tusur.ru/publications/6357 (дата обращения: 15.05.2018).
9. Панов, С. А. Теория и технологии программирования: Курс лекций [Электронный ресурс] / С. А. Панов – Томск: ТУСУР, 2015. – 116 с. – Режим доступа: https://edu.tusur.ru/publications/5013 (дата обращения: 15.05.2018).
10. Кручинин, В. В. Технологии программирования: Учебное пособие [Электронный ресурс] / В. В. Кручинин. – Томск: ТУСУР, 2013. – 271 с. – Режим доступа: https://edu.tusur.ru/publications/2834 (дата обращения: 15.05.2018).
11. Перегудов Ф.И., Тарасенко Ф.П. Основы системного анализа: Учеб. пособие. – 3-е изд. – Томск: Изд-во НТЛ, 2001. – 396 с.
12. Информатика : учебник для вузов / Н. В. Макарова, В. Б. Волков. – СПб. : ПИТЕР, 2012. – 576 с.
13. Информатика: базовый курс : учебник для вузов / О. А. Акулов, Н. В. Медведев. – 8-е изд., стереотип. – М. : Омега-Л, 2013. – 576 с.

14. Информатика. Базовый курс : Учебник для вузов / С. В. Симонович [и др.] ; ред. : С. В. Симонович. – 2-е изд. – СПб. : Питер, 2007. - 639[1] с.

15. C/C++. Программирование на языке высокого уровня : учебник для вузов / Т. А. Павловская. – СПб. : ПИТЕР, 2013. – 461 с.

16. Кручинин, В. В. Технологии программирования: Учебное пособие [Электронный ресурс] / В. В. Кручинин. – Томск: ТУСУР, 2013. – 271 с. – Режим доступа: https://edu.tusur.ru/publications/2834

17. Управление данными : учебник для вузов / А. В. Кузовкин, А. А. Цыганов, Б. А. Щукин. – М. : Академия, 2010. – 256 с.

18. Салмина Н. Ю., Функциональное программирование и интеллектуальные системы: учебное пособие [Электронный ресурс] / Салмина Н. Ю. – Томск: ТУСУР, 2016 . – 100 с. – Режим доступа: https://edu.tusur.ru/publications/6357 (дата обращения: 15.05.2018).

19. Представление знаний в информационных системах [Текст] : учебник для вузов / Б. Я. Советов, В. В. Цехановский, В. Д. Чертовской. – М. : Академия, 2011. – 144 с. : табл. – (Высшее профессиональное образование. Информатика и вычислительная техника). – Библиогр.: с. 140-142.

20. Кручинин, В. В. Профессиональные математические пакеты: Учебно-методическое пособие [Электронный ресурс] / В.В. Кручинин, М. Ю. Перминова. – Томск: ТУСУР, 2017. – 117 с. – Режим доступа: https://edu.tusur.ru/publications/7256 (дата обращения: 15.05.2018).

21. Ульянов М. В. Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ [Электронный ресурс] : учеб. пособие – Электрон. дан. – М. : Физматлит, 2008. – 304 с. – Режим доступа: https://e.lanbook.com/book/2354 (дата обращения: 15.05.2018).

22. Кремер Н.Ш. и др. Исследование операций в экономике. Учебное пособие для вузов/ ред. : Н. Ш. Кремер. – М. : ЮНИТИ, 2006. – 407 с.

23. Грибанова, Е. Б. Исследование операций и методы оптимизации : Учебное пособие [Электронный ресурс] / Грибанова Е. Б., Мицель А. А. – Томск: ТУСУР, 2017. – 185 с. – Режим доступа: https://edu.tusur.ru/publications/7127 (дата обращения: 15.05.2018).

24. Казарин О.В. Надежность и безопасность программного обеспечения [Электронный ресурс] : учеб. пособие / О.В. Казарин, И.Б. Шубинский. – М.: Издательство Юрайт, 2018. – 342 с. – Режим доступа: https://biblio-online.ru/book/6A637EC7-8B78-4DA6-B404-71DE0202E2EF (дата обращения: 24.04.2018).

25. Майерс Г. Искусство тестирования программ : Пер. с англ. / Гленфорд Дж. Майерс; Ред. пер. Б. А. Позин. – М. : Финансы и статистика, 1982. – 176 с.