

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ (ФИЛИАЛ)**

**ФЕДЕРАЛЬНОГО ГОСУДАРСТВЕННОГО БЮДЖЕТНОГО**

**ОБРАЗОВАТЕЛЬНОГО УЧРЕЖДЕНИЯ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

**В Г. ТАГАНРОГЕ РОСТОВСКОЙ ОБЛАСТИ**

**ПИ (филиал) ДГТУ в г. Таганроге**

УТВЕРЖДАЮ

Директор

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.К. Исаев

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2019 г

Рег. № \_\_\_\_\_\_\_\_\_\_\_\_

**УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ**

к практическим занятиям

По профессиональному модулю ПМ.02. Осуществление интеграции программных модулей

по специальности 09.02.07 «Информационные системы и программирование»

Таганрог

2020

**Лист согласования**

Учебно-методическое пособие по учебные дисциплины ***Осуществление интеграции программных модулей*** разработана на основе Федерального государственного образовательного стандарта (далее – ФГОС) для специальности среднего профессионального образования (далее – СПО) 09.02.07 «Информационные системы и программирование».

**Разработчик(и):**

Преподаватель Т.М. Марданова

Методические указания рассмотрены и одобрены на заседании цикловой методической комиссии специальности 09.02.05Прикладная информатика (по отраслям)

Протокол № 7 от «04» февраля 2020г

Председатель цикловой методической комиссии О.В. Андриян

**Рецензенты:**

ЧОУ ВО «ТИУиЭ» начальник информационно-аналитического управления, к.т.н., доцент О.И. Овчаренко

АО «Красный гидропресс»зам. начальника отдела ИТ С.С. Пирожков

**Согласовано:**

Заведующий УМО

Т. В. Воловская

**Введение**

В учебно-методическом пособии к практикуму по курсу «Осуществление интеграции программных модулей» изложены сведения, необходимые для успешного выполнения практических занятий по данному курсу. Описан процесс работы с инструментарием, применяемым на практических занятиях, представлен ряд типичных задач и подходы к их решению. Практические занятия посвящены углубленному знакомству обучающихся с разработкой требований к программным модулям на основе анализа проектной и технической документации на предмет взаимодействия компонент, выполнять интеграцию модулей в программное обеспечение, выполнять отладку программного модуля с использованием специализированных программных средств.

Цель настоящего пособия – помочь обучающимся при выполнении практических работ, выполняемых для закрепления знаний по теоретическим основам и получения практических навыков работы на компьютерах.

Обучающийся должен знать: модели процесса разработки программного обеспечения; основные принципы процесса разработки программного обеспечения; основные подходы к интегрированию программных модулей; основы верификации и аттестации программного обеспечения.

Обучающийся должен уметь: использовать выбранную систему контроля версий; использовать методы для получения кода с заданной функциональностью и степенью качества.

Данное учебно-методическое пособие предназначено для обучающихся 3 курса.

**Правила выполнения практических занятий**

Практические занятия выполняются каждым обучающимся самостоятельно в полном объеме и согласно содержанию методических указаний.

Перед выполнением обучающийся должен отчитаться перед преподавателем за выполнение предыдущего занятия (сдать отчет).

Обучающийся должен на уровне понимания и воспроизведения предварительно усвоить необходимую для выполнения практических занятий теоретическую и информацию.

Обучающийся, получивший положительную оценку и сдавший отчет по предыдущему практическому занятию, допускается к выполнению следующему занятию.

Обучающийся, пропустивший практическое занятие по уважительной либо неуважительной причине, закрывает задолженность в процессе выполнения последующих практических занятий.

**Практическое занятие №1 «Анализ предметной области»**

**Цель:**Изучить, описать и проанализировать предметную область, в которой будет создаваться информационная база.

**Форма отчета:**

1. Провести анализ предметной области в соответствии с выданным заданием.
2. Защитить практическую работу.

**Время выполнения:** 2 ч

**Выделяются следующие шаги работы над проектом (системой):**

1. Описание предметной области, под которой понимается та часть реального мира, которая имеет существенное значение или непосредственное отношение к процессу функционирования программы. Другими словами, предметная область включает в себя только те объекты и взаимосвязи между ними, которые необходимы для описания требований и условий решения некоторой задачи. Следовательно, разработчикам необходимо выделить основные объекты (компоненты), участвующие в функционировании системы, определить их наиболее существенные характеристики, взаимосвязи в рамках решаемой задачи, а также определить основные информационные потоки в системе. При этом отдельные компоненты выбираются таким образом, чтобы при последующей разработке их было удобно представить в форме классов и объектов. В этом случае немаловажное значение приобретает и сам язык представления информации о концептуальной схеме предметной области.

Сложность предметной области определяет количество объектов и связей между ними, поэтому описание должно включать в себя базовые *термины и определения*, сопровождаться различными примерами, в нем могут приводиться различного рода *классификации*, поясняющие различные свойства описываемых объектов. Если в системе используются математические модели, то они также должны быть описаны с учетом специфики применения.

2. Обзор существующих *систем-аналогов* – неотъемлемая часть описания предметной области, которая позволяет разработчику определить основные концепции, необходимые для реализации в системе. Описание должно приводиться с указанием отличительных особенностей разработанных систем, с перечислением их достоинств и недостатков, в отчете обязательно приводятся экранные формы этих систем.

3. Результатом последнего этапа является *диаграмма объектов предметной области*и краткое описание их свойств и функций. При построении данной диаграммы нужно помнить о том, что в данном случае объект – это «конкретная материализация абстракции», а не экземпляр класса. Диаграмма объектов представляет статическую составляющую взаимодействующих между собой объектов, она должна включить в себя только те объекты предметной области, которые потом преобразуются в диаграмму классов. Связи между объектами показывают отношения между ними, при необходимости в диаграмме можно привести и атрибуты (свойства) объектов.

Диаграммы объектов не позволяют полностью описать объектную структуру системы, поэтому при их использовании нужно сосредоточиться на изображении интересующих вас наборов конкретных объектов.

Для сбора, хранения, поиска и выдачи информации о предметной области и ее объектов настоящее время в информационных системах широко используются базы данных.

Анализ предметной области начинается с выделения сущностей и определения их свойств или атрибутов.

Видимые сущности представляют собой объекты предметной области, которые может распознать человек.

Поддерживаемые сущности или абстрактные сущности разрабатываются для физической поддержки общей логической модели.

**Практическое занятие № 2 «Разработка и оформление технического задания»**

**Цель работы:** Ознакомление с процедурой разработки технического задания на создание программного продукта с применением ГОСТ 19.102-77 «Стадии разработки программ и программной документации».

**Форма отчета:** выполнением данной лабораторной работы является правильно оформленное техническое задание

**Время выполнения: 2 ч**

**Техническое задание**

На данной стадии выполняются следующие работы:

1. Обоснование необходимости разработки программ:

- постановка задачи;

- сбор исходных материалов;

- выбор и обоснование критериев эффективности и качества;

- обоснование необходимости проведения научно-исследовательских работ.

2. Выполнение научно-исследовательских работ:

- определение структуры входных и выходных данных;

- предварительный выбор методов решения задач;

- обоснование целесообразности применения ранее разработанных программ;

- определение требований к техническим средствам;

- обоснование принципиальной возможности решения поставленных задач.

3. Разработка и утверждение технического задания:

- определение требований к программе;

- разработка технико-экономического обоснования разработки программы;

- определение стадий, этапов и сроков разработки программы и документации на нее;

- выбор языков программирования;

- определение необходимости проведения научно-исследовательской работы на последующих стадиях.

**Практическое занятие №3 «Построение архитектуры программного средства»**

**Цель работы:** Реализация начальных этапов процесса разработки программного средства в соответствии с ГОСТ Р ИСО/МЭК 12207.

**Форма отчета:**

С целью реализации начальных этапов разработки ПС в соответствии с техническим заданием:

−выполнить подготовительную работу;

−провести анализ требований к ПС;

−выполнить проектирование архитектуры ПС на высоком уровне.

**Время выполнения: 2 ч**

При возникновении потребностей в заказе, приобретении, разработке, эксплуатации и сопровождении программ перед всеми сторонами, вовлеченными в жизненный цикл программного средства (ПС), возникает целый ряд вопросов, связанных с определением и детальным структурированием жизненного цикла (ЖЦ) ПС, с организационными и техническими правами и обязанностями сторон, с управлением ЖЦ и контролем за его реализацией. Одним из действенных инструментов для решения данных вопросов является использование унифицированных подходов, закрепленных в современных международных и российских стандартах.

Понятия «жизненный цикл системы» или «жизненный цикл программного средства» часто появляются в статьях и звучат в разговорах разработчиков, по крайней мере, руководителей проектов и подразделений.

Всем понятно, что относятся они к тому, что и в какой последовательности должно делаться при создании и эксплуатации систем. Но прежде чем две организации или два специалиста договорятся о том, что конкретно входит или не входит в ЖЦ, проходит значительное время. А позже вполне может обнаружиться, что эти двое (две «стороны») все-таки по-разному понимают, какие работы будут входить в ЖЦ, а какие - нет, какие проверки будут планироваться, когда и т. д. Естественно, общие принципы организации работ описаны давно, но что делать сторонам в конкретном проекте — это каждый раз приходится решать заново.

В стандартах, регламентирующих жизненный цикл программных средств, обобщаются опыт и результаты исследований множества специалистов и рекомендуются наиболее эффективные современные методы и процессы создания и развития комплексов программ. В результате таких обобщений оттачиваются технологические процессы и приемы разработки, а также методическая база для их автоматизации.

ЖЦ ПС в стандартах представляет собой набор этапов, частных работ и операций в последовательности их выполнения и взаимосвязи, регламентирующих ведение работ от подготовки технического задания до завершения испытаний ряда версий и окончания эксплуатации ПС или информационной системы (ИС).

Стандарты включают правила описания исходной информации, способов и методов выполнения операций, устанавливают правила контроля технологических процессов, требования к оформлению их результатов, а также регламентируют содержание технологических и эксплуатационных документов на комплексы программ. Они определяют организационную структуру коллектива, обеспечивают распределение и планирование заданий, а также, контроль за ходом создания ПС.

Для того чтобы привнести порядок и понимание, общие для любых сторон, участвующих в ЖЦ систем и ПС, давно разрабатывались стандарты различных уровней утверждения - национальные и международные.

В России основы построения и использования профилей стандартов ЖЦ ПС заложены принятием в качестве базового стандарта ГОСТ Р ИСО/МЭК 12207. Данный документ введен в действие с 1 июля 2000 г., тесно взаимоувязан с рядом стандартов, принятых ранее, и с некоторыми стандартами, разрабатываемыми в данное время на основе прямого применения стандартов ИСО.

Актуальность стандарта ГОСТ Р ИСО/МЭК 12207 для современных условий настолько высока, что принятие в ISO его исходного, международного варианта вскоре вызвало самую положительную оценку российских экспертов. Был дан ряд рекомендаций по его использованию в реальных условиях.

В данном стандарте программное обеспечение (ПО) или программный продукт определяется как набор компьютерных программ*,*процедур исвязанной с ними документации и данных.

Процесс определяется как совокупность взаимосвязанных действий*,*преобразующих некоторые входные данные в выходные. Каждый процесс характеризуется определенными задачами и методами их решения, исходными данными, полученными от других процессов, и результатами.

В соответствии с ГОСТ Р ИСО/МЭК 12207 все процессы ЖЦ ПО разделены на три группы:

*1)*Основные процессы:

−приобретение;

−поставка;

−разработка;

−эксплуатация;

−сопровождение.

*2)*Вспомогательные процессы:

−документирование;

−управление конфигурацией;

−обеспечение качества;

−верификация;

−аттестация;

−совместная оценка;

−аудит;

−разрешение проблем.

*3)*Организационные процессы:

−управление;

−усовершенствование;

−создание инфраструктуры;

−обучение.

Процесс разработки предусматривает действия и задачи, выполняемые разработчиком, и включает следующие действия:

А) Подготовительная работа, которая начинается с выбора модели ЖЦ ПО, соответствующей масштабу, значимости и сложности проекта. Действия и задачи процесса должны соответствовать выбранной модели. Разработчик должен выбрать, адаптировать к условиям проекта и использовать согласованные с заказчиком стандарты, методы и средства разработки, а также составить план выполнения работ.

Б) Анализ требований к системе подразумевает определение ее функциональных возможностей, пользовательских требований, требований к надежности и безопасности, требований к внешним интерфейсам и т.д. Требования к системе оцениваются исходя из критериев реализуемости и возможности проверки при тестировании.

Анализ требований к ПО предполагает определение следующих характеристик для каждого компонента ПО:

−функциональных возможностей, включая характеристики производительности и среды функционирования компонента;

−внешних интерфейсов;

−спецификаций надежности и безопасности;

−эргономических требований;

−требований к используемым данным;

−требований к установке и приемке;

−требований к пользовательской документации;

−требований к эксплуатации и сопровождению.

Требования к ПО оцениваются исходя из критериев соответствия требованиям к системе, реализуемости и возможности проверки при тестировании.

В) Проектирование архитектуры системы на высоком уровне заключается в определении компонентов ее оборудования, ПО и операций, выполняемых эксплуатирующим систему персоналом. Архитектура системы должна соответствовать требованиям, предъявляемым к системе, а также принятым проектным стандартам и методам.

Проектирование архитектуры ПО включает следующие задачи:

− трансформацию требований к ПО в архитектуру, определяющую на высоком уровне структуру ПО и состав ее компонентов;

− разработку и документирование программных интерфейсов ПО и баз данных;

− разработку предварительной версии пользовательской документации;

− разработку и документирование предварительных требований к тестам и планам интеграции ПО.

Архитектура компонентов ПО должна соответствовать требованиям, предъявляемым к ним, а также принятым проектным стандартам и методам.

Г) Детальное проектирование ПО включает следующие задачи:

− описание компонентов и интерфейсов между ними на более низком уровне, достаточном для их последующего самостоятельного кодирования и тестирования;

− разработку и документирование детального проекта базы данных;

− обновление (при необходимости) пользовательской документации;

Д) Кодирование и тестирование ПО охватывает задачи:

− разработку и документирование каждого компонента ПО и базы данных, а также совокупности тестовых процедур и данных для их тестирования;

− тестирование каждого компонента ПО и базы данных на соответствие предъявляемых к ним требованиям. Результаты тестирования компонентов должны быть документированы;

− обновление (при необходимости) пользовательской документации;

− обновление плана интеграции ПО.

Е) Интеграция ПО предусматривает сборку разработанных компонентов ПО в соответствии с планом интеграции и тестирование агрегированных компонентов. Для каждого из агрегированных компонентов разрабатываются наборы тестов и тестовые процедуры, предназначенные для проверки каждого из квалификационных требований при последующем квалификационном тестировании.

Ж) Квалификационное тестирование - это набор критериев и условий,

которые необходимо выполнить, чтобы квалифицировать программный продукт как соответствующий своим спецификациям и готовый к использованию в условиях эксплуатации.

Квалификационное тестирование ПО проводится разработчиком, в присутствии заказчика (по возможности), для демонстрации того, что ПО удовлетворяет своим спецификациям и готово к использованию в условиях эксплуатации. Квалификационное тестирование выполняется для каждого компонента ПО по всем разделам требований при широком варьировании тестов.

З) Установка ПО осуществляется разработчиком в соответствии с планом в той среде и на том оборудовании, которые предусмотрены договором. В процессе установки проверяется работоспособность ПО и баз данных.

И) Приемка ПО предусматривает оценку результатов квалификационного тестирования ПО и системы и документирование результатов оценки, которые проводятся заказчиком с помощью разработчика.

**Задание:**разработать проект архитектуры программного средства в соответствии с ГОСТ Р ИСО/МЭК 12207.

**Практическое занятие №4 «Изучение работы в системе контроля версий»**

**Цель работы:**получение первоначальных навыков использования систем контроля версий исходного кода программ и первоначальных навыков организации коллективной разработки программного обеспечения.

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

Software Configuration Management или Конфигурационное управление подразумевает под собой комплекс методов, направленных на то, чтобы систематизировать изменения, вносимые разработчиками в программный продукт в процессе его разработки и сопровождения, сохранить целостность системы после изменений, предотвратить нежелательные и непредсказуемые эффекты, а также сделать процесс внесения изменений более формальным.

К процедурам можно отнести создание резервных копий, контроль исходного кода, требований проекта, документации и т. д. Степень формальности выполнения данных процедур зависит от размеров проекта, и при правильной ее оценке данная концепция может быть очень полезна. Конфигурационное управление требует выполнения множества трудоемких рутинных операций. На практике, в большинстве случаев, для конфигурационного управления применяются специальные системы контроля версий исходного кода программ. В качестве примера рассмотрим информационную систему Subversion. Это бесплатная система управления версиями с открытым исходным кодом.

Subversion позволяет управлять файлами и каталогами, а так же сделанными в них изменениями во времени. Это позволяет восстановить более ранние версии данных, предоставляет возможность изучить историю всех изменений.

**Задание:** Создать тестовый проект на любом знакомом языке программирования и отработать навыки использования хранилища на локальном компьютере.

**Практическое занятие № 5 «Построение диаграммы Вариантов использования и диаграммы Последовательности»**

**Цель работы:** научиться строить диаграмму вариантов использования.

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

**Теоретическая справка:**

Визуальное моделирование в UML можно представить как некоторый процесс поуровневого спуска от наиболее обшей и абстрактной концептуальной модели исходной системы к логической, а затем и к физической модели соответствующей программной системы. Для достижения этих целей вначале строится модель в форме так называемой диаграммы вариантов использования (use case diagram), которая описывает функциональное назначение системы или, другими словами, то, что система будет делать в процессе своего функционирования. Диаграмма вариантов использования является исходным концептуальным представлением или концептуальной моделью системы в процессе ее проектирования и разработки.

Разработка диаграммы вариантов использования преследует цели:

* Определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы.
* Сформулировать общие требования к функциональному поведению проектируемой системы.
* Разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей.
* Подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью так называемых вариантов использования. При этом актером (actor) или действующим лицом называется любая сущность, взаимодействующая с системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик. В свою очередь, вариант использования (use case) служит для описания сервисов, которые система предоставляет актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемый системой при диалоге с актером. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие актеров с системой.

**Состав диаграммы Use Case**

Диаграмма вариантов использования состоит из актеров, для которых система производит действие и собственно действия Use Case, которое описывает то, что актер хочет получить от системы. Актер обозначается значком человечка, а Use Case - овалом. Дополнительно в диаграммы могут быть добавлены комментарии.

**Виды взаимодействий**

Между актерами и вариантами использования могут быть различные виды взаимодействия. Основные виды взаимодействия следующие:

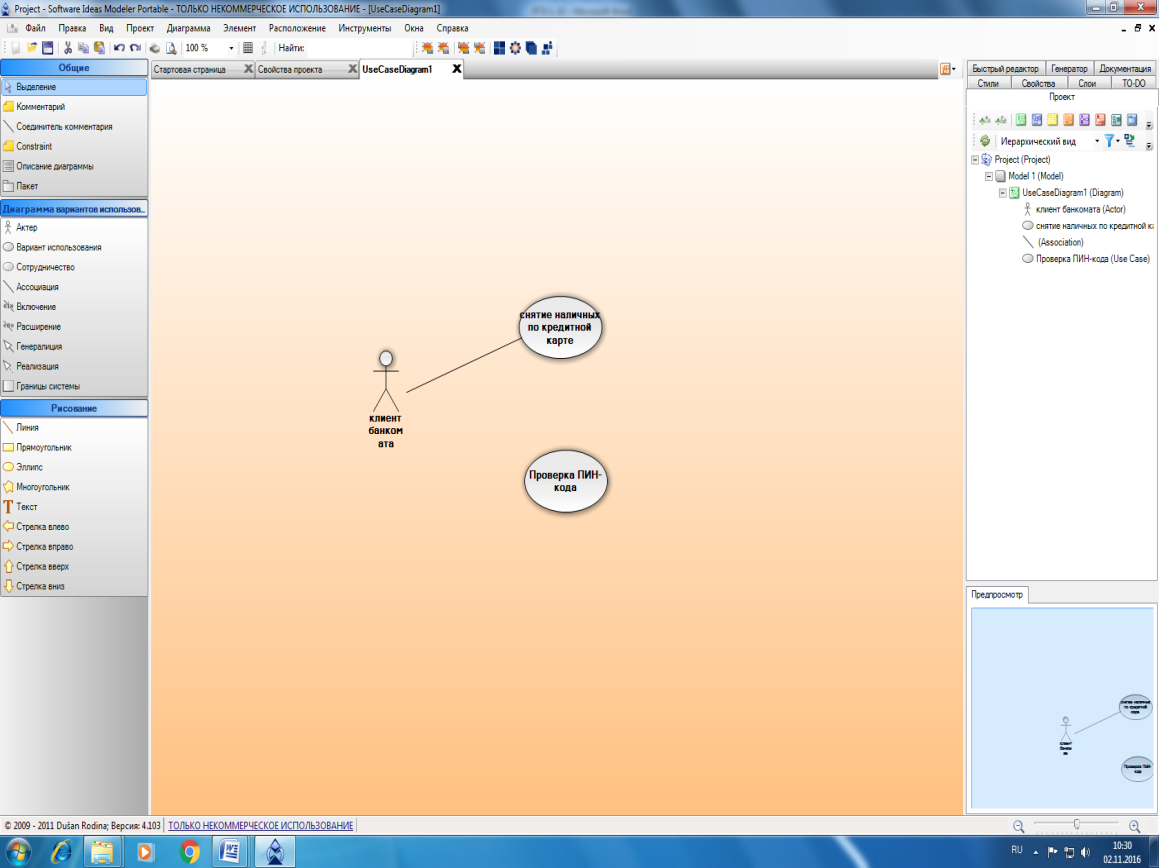
* **Простая ассоциация** - отражается линией между актером и вариантом использования (без стрелки). Отражает связь актера и варианта использования. На рисунке между актером администратор и вариантом использованияпросматривать заказ.
* **Направленная ассоциация**- то же что и простая ассоциация, но показывает, что вариант использования инициализируется актером. Обозначается стрелкой.
* **Наследование**- показывает, что потомок наследует атрибуты и поведение своего прямого предка. Может применяться как для актеров, так для вариантов использования.
* **Расширение**(extend) - показывает, что вариант использования расширяет базовую последовательность действий и вставляет собственную последовательность. При этом в отличие от типа отношений "включение" расширенная последовательность может осуществляться в зависимости от определенных условий.
* **Включение**(include)- показывает, что вариант использования включается в базовую последовательность и выполняется всегда (на рисунке не показан).

Существуют и другие виды взаимодействия, но они менее важны и реже применяются.

**Задание 1.**Построить диаграмму вариантов использования модели вариантов использования банкомата.

Выполните следующие действия:

1. Добавить актера с именем Клиент банкомата.
2. Добавить вариант использования Снятие наличных по кредитной карте
3. Добавить направленную ассоциацию от бизнес-актера Клиент Банкомата к варианту использования Снятие наличных по кредитной карте
4. Добавить вариант использования Проверка ПИН-кода.



1. Добавить актера с именем Банк.
2. Добавить вариант использования Получение справки о состоянии счета.
3. Добавить вариант использования Блокирование кредитной карточки.
4. Добавить направленную ассоциацию от бизнес-актера Клиент Банкомата к варианту использования Получение справки о состоянии счета.
5. Добавить направленную ассоциацию от варианта использования Снятие наличных по кредитной карточке к сервису Банк.
6. Добавить направленную ассоциацию от варианта использования Получение справки о состоянии счета к сервису Банк.
7. Добавить отношение зависимости со стереотипом «include», направленное от варианта использования Снятие наличных по кредитной карте к варианту использования Проверка Пин-кода.
8. Добавить отношение зависимости со стереотипом «include», направленное от варианта использования Получение справки о состоянии счета к варианту использования Проверка Пин-кода.
9. Добавить отношение зависимости со стереотипом «extend», направленное от варианта использования Блокирование кредитной карточки к варианту использования Проверка Пин-кода.

**Задание 2.** Построить диаграмму вариантов использования.

Имеются следующие данные:

* четыре действующих лица: Клиента банка, Банк, Кассира и Оператора,
* пять вариантов использования: Снять наличные, Перевести деньги со счета, Положить деньги на счет, Пополнить запас денег и Подтвердить пользователя,
* три зависимости, и отношения между действующими лицами и вариантами использования.

Варианты использования: Снять наличные, Перевести деньги со счета, Положить деньги на счет - требуют включения идентификации клиента в системе. Это поведение может быть выделено в новый вариант использования включения, называемый Подтвердить пользователя. Базовые варианты использования не зависимы от метода, используемого для идентификации. Поэтому он инкапсулируется (скрывается) в варианте использования включения. С точки зрения базовых вариантов использования не имеет значение производится ли идентификация с помощью магнитной карты или сканированием сетчатки глаза. Они только зависят от результата выполнения варианта использования Подтвердить клиента.

**Задание для самостоятельной работы:**Построить диаграмму вариантов использования на основе вербальной модели информационной системы «Компьютерный клуб»

**Контрольные вопросы:**

1. Какие цели преследует разработка диаграммы использования?
2. Для чего нужна диаграмма вариантов использования?
3. Из чего состоит диаграмма вариантов использования?
4. Виды взаимодействия используемые в диаграмме вариантов использования?
5. Из чего состоит созданная вами диаграмма?

**Практическое занятие № 6 «Построение диаграммы Кооперации и диаграммы Развертывания»**

**Цель:** ознакомиться с методологией моделирования информационных систем на основе языка UML. Теоретические вопросы Универсальный язык моделирования UML. Понятие диаграммы. Виды диаграмм. Основные элементы диаграммы кооперации. Основные элементы диаграммы развертывания.

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

Задание № 1. Ознакомиться с методологией построения диаграммы кооперации основе языка UML. Задание № 2. Проанализируйте пример построения диаграммы кооперации (рисунок 4).

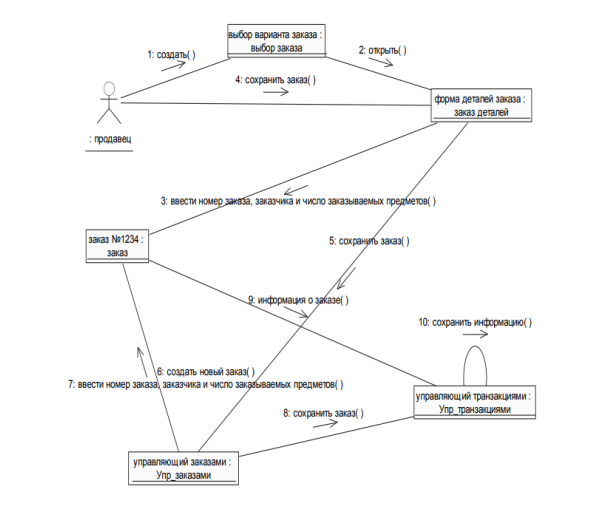


Рис. 4

Задание № 3. Постройте диаграмму кооперации для выбранной информационной системы (практическая работа № 11). Задание № 4. Ознакомиться с методологией построения диаграммы развертывания основе языка UML. Задание № 5. Проанализируйте пример построения диаграммы развертывания. Примеры построения диаграмм развертывания Фрагмент диаграммы развертывания с соединениями между узлами показан на рисунке 5.

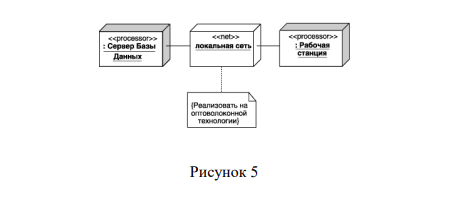


Диаграмма развертывания с отношением зависимости между узлом и развернутыми на нем компонентами приведена на рисунке 6.

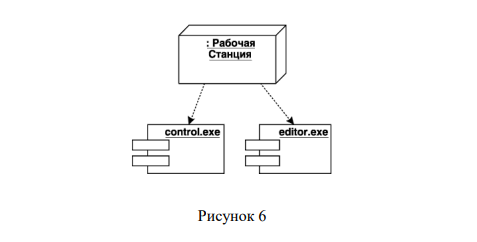
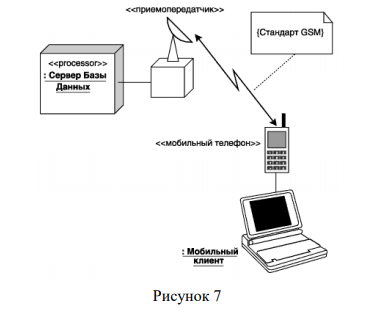


Диаграмма развертывания для системы мобильного доступа к корпоративной базе данных изображена на рисунке 7.



Задание № 6. Постройте диаграмму развертывания для выбранной информационной системы (практическая работа №11). Задание № 7. Оформите отчет.

**Практическое занятие № 7 «Построение диаграммы Деятельности, диаграммы Состояний и диаграммы Классов»**

**Целью работы** является изучение основ создания диаграмм деятельности на языке UML, получение навыков построения диаграмм деятельности, применение приобретенных навыков для построения объектноориентированных моделей определенной предметной области.

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

Задачи Основными задачами практической работы являются: ‒ ознакомиться с теоретическими вопросами построения диаграмм деятельности на языке UML; ‒ ознакомиться с теоретическими вопросами построения диаграмм деятельности с помощью MS Visio. 3. Краткие теоретические сведения При моделировании поведения проектируемой или анализируемой системы возникает необходимость не только представить процесс изменения ее состояний, но и детализировать особенности алгоритмической и логической реализации выполняемых системой операций. Традиционно для этой цели использовались блок-схемы или структурные схемы алгоритмов. Каждая такая схема акцентирует внимание на последовательности выполнения определенных действий или элементарных операций, которые в совокупности приводят к получению желаемого результата. Алгоритмические и логические операции, требующие выполнения в определенной последовательности, окружают нас постоянно. Например, чтобы позвонить по телефону, нам предварительно нужно снять трубку или включить его. Для приготовления кофе или заваривания чая необходимо вначале вскипятить воду. Чтобы выполнить ремонт двигателя автомобиля, требуется осуществить целый ряд нетривиальных операций, таких как разборка силового агрегата, снятие генератора и некоторых других. C увеличением сложности системы строгое соблюдение последовательности выполняемых операций приобретает все более важное значение. Если попытаться заварить кофе холодной водой, то мы можем только испортить одну порцию напитка. Нарушение последовательности операций при ремонте двигателя может привести к его поломке или выходу из строя. Еще более катастрофические последствия могут произойти в случае отклонения от установленной последовательности действий при взлете или посадке авиалайнера, запуске ракеты, регламентных работах на АЭС.

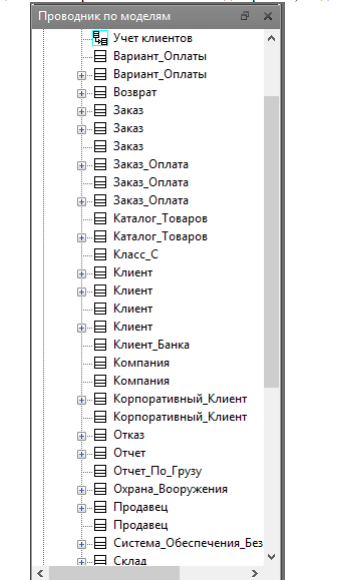
Для моделирования процесса выполнения операций в языке UML используются так называемые диаграммы деятельности. Применяемая в них графическая нотация во многом похожа на нотацию диаграммы состояний, поскольку на диаграммах деятельности также присутствуют обозначения состояний и переходов. Отличие заключается в семантике состояний, которые используются для представления не деятельностей, а действий, и в отсутствии на переходах сигнатуры событий. Каждое состояние на диаграмме деятельности соответствует выполнению некоторой элементарной операции, а переход в следующее состояние срабатывает только при завершении этой операции в предыдущем состоянии. Графически диаграмма деятельности представляется в форме графа деятельности, вершинами которого являются состояния действия, а дугами – переходы от одного состояния действия к другому. Таким образом, диаграммы деятельности можно считать частным случаем диаграмм состояний. Основным направлением использования диаграмм деятельности является визуализация особенностей реализации операций классов, когда необходимо представить алгоритмы их выполнения. При этом каждое состояние может являться выполнением операции некоторого класса либо ее части, позволяя использовать диаграммы деятельности для описания реакций на внутренние события системы. В контексте языка UML деятельность (activity) представляет собой некоторую совокупность отдельных вычислений, выполняемых автоматом. При этом отдельные элементарные вычисления могут приводить к некоторому результату или действию (action). На диаграмме деятельности отображается логика или последовательность перехода от одной деятельности к другой, при этом внимание фиксируется на результате деятельности. Сам же результат может привести к изменению состояния системы или возвращению некоторого значения. Состояние действия и деятельности Состояние деятельности (activity state) – состояние в графе деятельности, которое служит для представления процедурной последовательности действий, требующих определенного времени. Переход из состояния деятельности происходит после выполнения специфицированной в нем ду-деятельности, при этом ключевое слово do в имени деятельности не указывается. Состояние деятельности не может иметь внутренних переходов, поскольку оно является элементарным.

**Методика выполнения** В качестве примера рассматривается моделирование системы продажи товаров по каталогу.

1. Запустите MS Visio.

2. Откройте файл, созданный в практических занятиях №8-9 и содержащий диаграмму классов.

3. В проводнике по моделям должны отображаться все классы и диаграммы, созданные ранее (рис. 9).



Опишем с помощью диаграммы деятельности процесс формирования заказа и выдачу товара. В бизнеспроцессе участвуют 3 действующих лица: клиент, продавец и система оплаты. Следовательно, необходимо добавить 3 дорожки для распределения ответственности между этими лицами. Для этого, в файле с диаграммой классов, созданной в практическом занятии 8, необходимо проделать следующие действия:

6. Щелкнуть правой кнопкой мыши по классу Заказ.

7. В контекстном меню выбрать пункт Схемы.

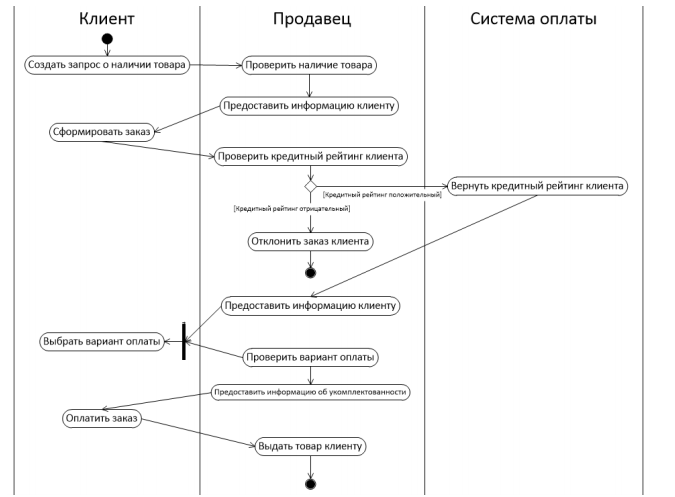
8. Нажать кнопку Создать и выбрать Деятельность.

9. Переименовать созданный лист в Деятельность-Заказ.

10.Построить диаграмму деятельности для класса Заказ. Для это выполните действия, описанные ниже.

a. Добавить 3 элемента Дорожка и изменить их названия на Клиент, Продавец и Система оплаты соответственно.

b. Добавить элементы Начальное состояние, Конечное состояние, Состояние действия, Решение, Переход (объединение), изменить их названия и задать расположение в соответствии с рисунком 10.



Задание Построить диаграмму деятельности в соответствии с вариантом. Отчет по практическому занятию выполняется в формате MS Word, который содержит пошаговое описание процесса построения диаграммы, а также скриншоты результатов согласно заданию. Создать диаграммы деятельности не менее чем для трех классов, описанных в практическом занятии №8.

Варианты 1. «Отдел кадров»;

2. «Агентство аренды»;

3. «Аптека»;

4. «Ателье»;

5. «Аэропорт»;

6. «Библиотека»;

7. «Кинотеатр»;

8. «Поликлиника»;

9. «Автосалон»;

10.«Таксопарк».

**Практическое занятие № 8 «Построение диаграммы компонентов»**

Цель и содержание работы: научиться создавать диаграммы Компонентов системы, добавлять компоненты к пакетам, изображать зависимости.

Постановка задачи

Разработаем диаграммы Компонентов и создадим для каждого класса соответствующие языку программирования С++ компоненты.

Теоретическое обоснование

Диаграмма компонентов предназначена для распределения классов и объектов по компонентам при физическом проектировании системы. Компоненты на диаграмме компонентов представляют собой физиче­ские модули программного кода. Обычно они в точности соот­ветствуют пакетам на диаграмме пакетов; таким образом, диаграмма компонентов отражает выполнение каждого пакета в системе.

Зависимости между компонентами должны совпадать с зависимостями между пакетами. Эти зависимости показывают, каким образом одни ком­поненты взаимодействуют с другими. Направление данной зависимости показывает уровень осведомленности о коммуникации.

Методика и порядок [выполнения работы](https://pandia.ru/text/category/vipolnenie_rabot/)

На рис.34 показана главная диаграмма Компонентов всей системы. Внимание на ней уделяется пакетам создаваемых компонентов.

На рис.35 показаны компоненты пакета Entities. Эти компоненты содержат классы пакета Entities Логического представления системы.

На рис.36 показаны компоненты пакета Control. Они содержат классы пакета Control Логического представления системы.

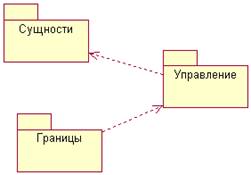


Рис.34 - Главная диаграмма Компонентов системы

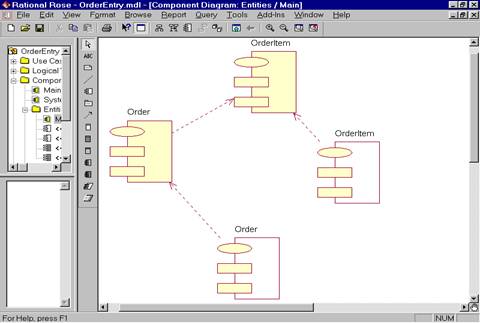


Рис.35 - Диаграмма Компонентов пакета Entities

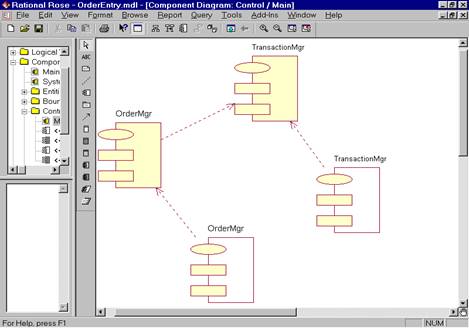


Рис.36 - Диаграмма Компонентов пакета Control

Наконец, на рис.37 показаны компоненты пакета Boundaries. Они также соответствуют классам одноименного пакета Логического представления системы.

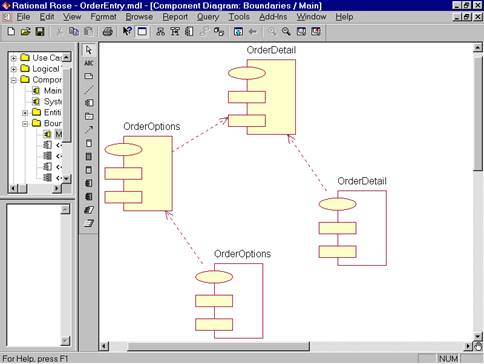


Рис.37 - Диаграмма Компонентов пакета Boundaries

На рис.38 показаны диаграмма Компонентов системы. На ней вы можете видеть все зависимости между всеми компонентами проектируемой системы.

## Этапы выполнения упражнения

### Создание пакетов компонентов

1.  Щелкните правой кнопкой мыши на представлении компонентов в броузере.

2.  В открывшемся меню выберите пункт New > Package (Создать > пакет).

3.  Назовите этот пакет Entities (Сущности).

4.  Повторите этапы с первого по третий, создав пакеты Boundaries (Границы) и Control (Управление).

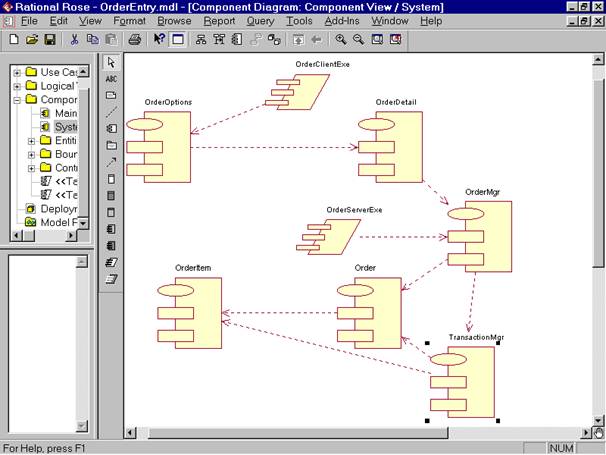


Рис.38 - Диаграмма Компонентов системы.

### Добавление пакетов на Главную диаграмму Компонентов

1.  Откройте Главную диаграмму Компонентов, дважды щелкнув на ней.

2.  Перетащите пакеты Entities, Boundary и Control из броузера на Главную диаграмму.

### Рисование зависимостей между пакетами

1.  На панели инструментов нажмите кнопку Dependency (Зависимость).

2.  Щелкните мышью на упаковке Boundaries Главной диаграммы Компонентов.

3.  Проведите линию зависимости до упаковки Control.

4.  Повторите этапы 1 - 3, проведя еще зависимость от пакета Control до пакета Entities.

### Добавление компонентов к пакетам и рисование зависимостей

1.  Дважды щелкните мышью на пакете Entities Главной диаграммы Компонентов, открыв Главную диаграмму Компонентов этого пакета.

2.  На панели инструментов нажмите кнопку Package Specification (Спецификация пакета).

3.  Поместите спецификацию пакета на диаграмму.

4.  Введите имя спецификации пакета OrderItem.

5.  Повторите этапы 2 - 4, добавив спецификацию пакета Order.

6.  На панели инструментов нажмите кнопку Package Body (Тело пакета).

7.  Поместите его на диаграмму.

8.  Введите имя тела пакета OrderItem.

9.  Повторите этапы 6 - 8, добавив тело пакета Order.

10.  На панели инструментов нажмите кнопку Dependency (Зависимость).

11.  Щелкните мышью на теле пакета OrderItem.

12.  Проведите линию зависимости от него к спецификации пакета OrderItem.

13.  Повторите этапы, добавив линию зависимости между телом пакета Order и спецификацией пакета Order.

14.  Повторите этапы, добавив линию зависимости от спецификации пакета Order к спецификации пакета OrderItem.

15.  С помощью описанного метода создайте следующие компоненты и зависимости:

Для пакета Boundaries:

# Спецификацию пакета OrderOptions

# Тело пакета OrderOptions

# Спецификацию пакета OrderDetail

# Тело пакета OrderDetail

Зависимости в пакете Boundaries:

# От тела пакета OrderOptions до спецификации пакета OrderOptions

# От тела пакета OrderDetail до спецификации пакета OrderDetail

# От спецификации пакета OrderOptions до спецификации пакета OrderDetail

### Создание диаграммы Компонентов системы

1.  Щелкните правой кнопкой мыши на представлении Компонентов в броузере.

2.  В открывшемся меню выберите пункт New→Component Diagram

3.  Назовите новую диаграмму System.

4.  Дважды щелкните на этой диаграмме.

### Размещение компонентов на диаграмме Компонентов системы

1.  Если это еще не было сделано, разверните в броузере пакет компонентов Entities, чтобы открыть его.

2.  Щелкните мышью на спецификации пакета Order в пакете компонентов Entities.

3.  Перетащите эту спецификацию на диаграмму.

4.  Повторите этапы 2 и 3, поместив на диаграмму спецификацию пакета OrderItem.

5.  С помощью этого метода поместите на диаграмму следующие компоненты:

Из пакета компонентов Boundaries:

# Спецификацию пакета OrderOptions

# Спецификацию пакета OrderDetail

Из пакета компонентов Control:

# Спецификацию пакета OrderMgr

# Спецификацию пакета TransactionMgr

6.  На панели инструментов нажмите кнопку Task Specification (Спецификация задачи).

7.  Поместите спецификацию задачи на диаграмму и назовите ее OrderClientExe.

8.  Повторите этапы 6 и 7 для спецификации задачи OrderServerExe.

Оформите отчет в виде скриншотов в ворде.

**Практическое занятие № 9 «Построение диаграмм потоков данных»**

**Цель работы**

Ознакомиться с методологией построения диаграмм потоков данных с использованием программного продукта BpWin.

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

**Содержание работы и методические указания к ее выполнению**

Диаграммы потоков данных (Data Flow Diagrams - DFD) используются для описания движения документов и обработки информации как дополнение к IDEF0. В отличие от IDEF0, где система рассматривается как взаимосвязанные работы, стрелки в DFD показывают лишь то, как объекты (включая данные) движутся от одной работы к другой. DFD отражает функциональные зависимости значений, вычисляемых в системе, включая входные значения, выходные значения и внутренние хранилища данных. DFD - это граф, на котором показано движение значений данных от их источников через преобразующие их процессы к их потребителям в других объектах.

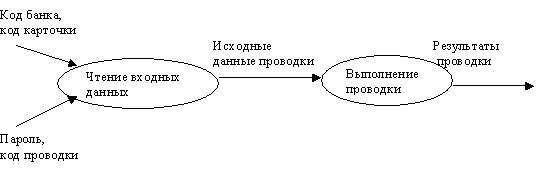
DFD содержит процессы, которые преобразуют данные, потоки данных, которые переносят данные, активные объекты, которые производят и потребляют данные, и хранилища данных, которые пассивно хранят данные.

Диаграмма потоков данных содержит:

* процессы, которые преобразуют данные;
* потоки данных, переносящие данные;
* активные объекты, которые производят и потребляют данные;
* хранилища данных, которые пассивно хранят данные.

Процесс DFD преобразует значения данных и изображается в виде эллипса, внутри которого помещается имя процесса.

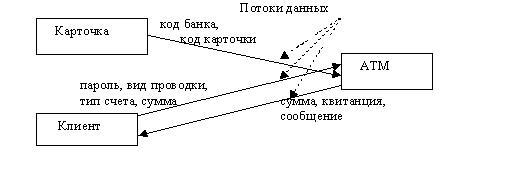
<="" p="">



Поток данных соединяет выход объекта (или процесса) с входом другого объекта (или процесса) и представляет собой промежуточные данные вычислений. Поток данных изображается в виде стрелки между производителем и потребителем данных, помеченной именами соответствующих данных. Дуги могут разветвляться или сливаться, что означает соответственно разделение потока данных на части либо слияние объектов.

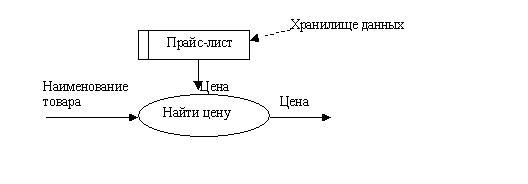
Активным объектом является объект, который обеспечивает движение данных, поставляя или потребляя их. Хранилище данных - это пассивный объект в составе DFD, в котором данные сохраняются для последующего доступа.

<="" p="">



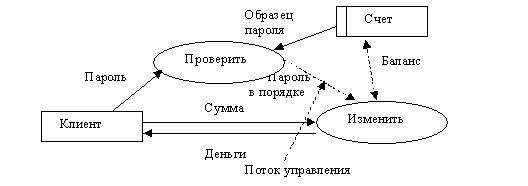
**Хранилища данных.** Хранилище данных - это пассивный объект в составе DFD, в котором данные сохраняются для последующего доступа. Хранилище данных допускает доступ к хранимым в нем данным в порядке, отличном от того, в котором они были туда помещены. Агрегатные хранилища данных, как, например, списки и таблицы, обеспечивают доступ к данным в порядке их поступления, либо по ключам.

<="" p="">



**Потоки управления.**DFD показывает все пути вычисления значений, но не показывает в каком порядке значения вычисляются. Решения о порядке вычислений связаны с управлением программой, которое отражается в динамической модели. Эти решения, вырабатываемые специальными функциями, или предикатами, определяют, будет ли выполнен тот или иной процесс, но при этом не передают процессу никаких данных, так что их включение в функциональную модель необязательно. Тем не менее, иногда бывает полезно включать указанные предикаты в функциональную модель, чтобы в ней были отражены условия выполнения соответствующего процесса.Функция, принимающая решение о запуске процесса, будучи включенной в DFD, порождает в диаграмме поток управления и изображается пунктирной стрелкой.

<="" p="">



Первым шагом при построении иерархии DFD является построение контекстных диаграмм. Обычно при проектировании относительно простых информационных систем строится единственная контекстная диаграмма со звездообразной топологией, в центре которой находится так называемый главный процесс, соединенный с приемниками и источниками информации, посредством которых с системой взаимодействуют пользователи и другие внешние системы.

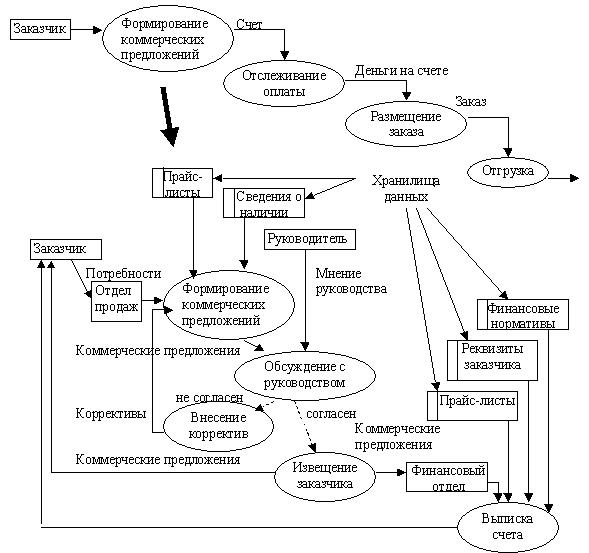
Если же для сложной системы ограничиться единственной контекстной диаграммой, то она будет содержать слишком большое количество источников и приемников информации, которые трудно расположить на листе бумаги нормального формата, и, кроме того, главный единственный процесс не раскрывает структуры распределенной системы.

Для сложных информационных систем строится иерархия контекстных диаграмм. При этом контекстная диаграмма верхнего уровня содержит не главный единственный процесс, а набор подсистем, соединенных потоками данных. Контекстные диаграммы следующего уровня детализируют контекст и структуру подсистем.

При построении иерархии DFD переходить к детализации процессов следует только после определения содержания всех потоков и накопителей данных, которое описывается при помощи структур данных. Структуры данных конструируются из элементов данных и могут содержать альтернативы, условные вхождения и итерации. Условное вхождение означает, что данный компонент может отсутствовать в структуре. Альтернатива означает, что в структуру может входить один из перечисленных элементов. Итерация означает вхождение любого числа элементов в указанном диапазоне. Для каждого элемента данных может указываться его тип (непрерывные или дискретные данные). Для непрерывных данных может указываться единица измерения (кг, см и т.п.), диапазон значений, точность представления и форма физического кодирования. Для дискретных данных может указываться таблица допустимых значений.

Ниже приведена диаграмма потоков данных верхнего уровня с ее последующим уточнением:

<="" p="">



Выполнение лабораторной работы производится с использованием программного продукта Enterprise Architect (методические указания по использованию Enterprise Architect)

**Последовательность выполнения лабораторной работы:**

1. Ознакомиться с методологией диаграмм потоков данных.

2. Ознакомиться с программным продуктом Enterprise Architect в части средств работы с диаграммами потоков данных.

3. Построить серию диаграмм потоков данных для отдельных сценариев работ, отражающих логику и взаимоотношение подразделений (подсистем).

4. Оформить отчет.

**Практическое занятие 10 «Разработка тестового сценария»**

**Цель:** получить навыки разработки тестовых сценариев. Теоретические вопросы Оценка стоимости и причины ошибок в программном обеспечении. Виды и методы тестирования. Понятие теста. Требования к разработке тестовых сценариев. Правила разработки тестовых сценариев.

**Форма отчета:**

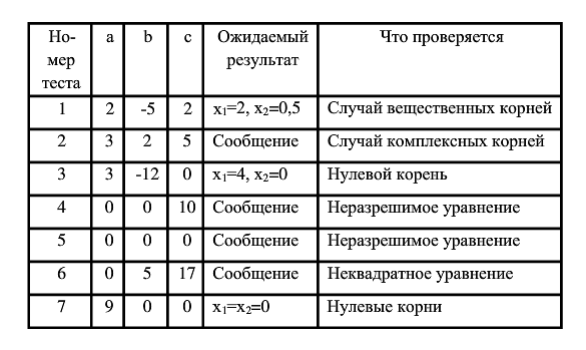
−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

Задание № 1. Написать программу решения квадратного уравнения ах2 + bх + с = 0. Задание № 2. Найти минимальный набор тестов для программы нахождения веще-ственных корней квадратного уравнения ах2 + bх + с = 0. Решение представлено в таблице.



Таким образом, для этой программы предлагается минимальный набор функциональных тестов, исходя из 7 классов выходных данных. Заповеди по отладки программного средства, предложенные Г. Майерсом.

Заповедь 1. Считайте тестирование ключевой задачей разработки ПС, поручайте его самым квалифицированным и одаренным программистам, нежелательно тестировать свою собственную программу.

Заповедь 2. Хорош тот тест, для которого высока вероятность обнаружить ошибку, а не тот, который демонстрирует правильную работу программы.

Заповедь 3. Готовьте тесты как для правильных, так и для неправильных данных.

Заповедь 4. Документируйте пропуск тестов через компьютер, детально изучайте результаты каждого теста, избегайте тестов, пропуск которых нельзя повторить.

Заповедь 5. Каждый модуль подключайте к программе только один раз, никогда не изменяйте программу, чтобы облегчить ее тестирование.

Заповедь 6. Пропускайте заново все тесты, связанные с проверкой работы какой-либо программы ПС или ее взаимодействия с другими программами, если в нее были внесены изменения (например, в результате устранения ошибки).

Задание № 6. Разработайте набор тестовых сценариев (как позитивных, так и негативных) для следующей программы: Имеется консольное приложение (разработайте самостоятельно). Ему на вход подается 2 строки. На выходе приложение выдает число вхождений второй строки в первую. Например:



Набор тестовых сценариев запишите в виде таблицы, приведенной выше.

Задание № 3. Оформите отчет.

**Практическое занятие № 12 «Оценка необходимого количества тестов»**

**Цель:** Научиться проводить оценку качества программного средства по различным показателям.

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

**Основные теоретические сведения**

Все программы по характеру использования и категориям пользователей можно разделить на два класса - утилитарные программы и программные продукты (изделия). Утилитарные программы («программы для себя») предназначены для удовлетворения нужд их разработчиков. Чаще всего утилитарные программы выполняют роль сервиса в технологии обработки данных либо являются программами решения функциональных задач, не предназначенных для широкого распространения. Программные продукты (изделия) предназначены для удовлетворения потребностей пользователей, широкого распространения и продажи. Существуют и другие варианты легального распространения программных продуктов: - freeware – бесплатные программы, свободно распространяемые, поддерживаются самим пользователем, который правомочен вносить в них необходимые изменения; - shareware – некоммерческие (условно-бесплатные) программы, которые могут использоваться, как правило, бесплатно. Ряд производителей использует ОЕМ - программы (Original Equipment Manufacturer), т.е. встроенные программы, устанавливаемые на компьютеры или поставляемые вместе с вычислительной техникой. Программные продукты (ПП) могут создаваться как: - индивидуальная разработка под заказ; - разработка для массового распространения среди пользователей.

Основными характеристиками программ являются:

• алгоритмическая сложность (логика алгоритмов обработки информации);

• состав и глубина проработки реализованных функций обработки;

• полнота и системность функций обработки;

• объём файлов программ;

• требования к операционной системе и техническим средствам обработки со стороны программного средства;

• объём дисковой памяти;

• размер оперативной памяти для запуска программ;

• тип процессора;

• версия операционной системы;

• наличие вычислительной сети и др. Программные продукты имеют многообразие показателей качества, которые отражают различные аспекты. Основная характеристика программного продукта – это его общая полезность, которая включает в себя мобильность, исходную полезность и удобство эксплуатации. Мобильность ПП означает их независимость от технического комплекса системы обработки данных, операционной среды, сетевой технологии обработки данных, специфики предметной области и т.п. Мобильный (многоплатфорный) программный продукт может быть установлен на различных моделях компьютеров и операционных систем, без ограничений на его эксплуатацию в условиях вычислительной сети. Функции обработки такого программного продукта для массового использования без каких-либо изменений. Исходная полезность характеризуется следующими показателями: - надежность;

- эффективность;

- учет человеческого фактора;

Надежность работы ПП определяется бессбойностью и устойчивостью в работе программ, точностью выполнения предписанных функций обработки, возможностью диагностики возникающих в процессе работы программ ошибок. Эффективность ПП оценивается как с позиций прямого его назначения – требований пользователя, так и сточки зрения расхода вычислительных ресурсов, необходимых для его эксплуатации. Расход вычислительных ресурсов оценивается через объем внешней памяти для размещения программ и объём оперативной памяти для запуска программ. Учёт человеческого фактора означает обеспечение дружественного интерфейса для работы конечного пользователя, наличие котекстно- зависимой подсказки или обучающей системы в составе программного средства, хорошей документации для освоения и использования, заложенных в программном средстве функциональных возможностей, анализ и диагностику возникших ошибок и др. Удобство эксплуатации включает следующие показатели качества: - модифицируемость; - коммуникативность. Модифицируемость ПП означает способность к внесению изменений, например расширение функций обработки, переход на другую техническую базу обработки и т.п. Коммуникативность ПП основана на максимально возможной их интеграции с другими программами, обеспечении обмене данными в общих форматах представления (экспорт/импорт баз данных, внедрение или связывание объектов обработки и др.). Естественно, что в условиях существования рынка программных продуктов важными характеристиками являются: стоимость; количество продаж; длительность продаж (время нахождения на рынке); известность фирмы-разработчика и программы; наличие программных продуктов аналогического назначения. Для оценки качества программного средства (ПС) используются различные способы получения информации о нём: - измерительный – основан на получении информации о свойствах и характеристиках ПС с использованием инструментальных средств (например, объём ПС); - регистрационный – получение информации во время испытаний или функционирования ПС, когда регистрируется и подсчитываются определённые события (число сбоев и отказов и др.);

Задание на лабораторную работу:

1. Скачать калькулятор любого производителя или взять разработанный студентами.

2. Сравнить два программных продукта: калькулятор фирмы Microsoft и калькулятор, написанный студентами (скачанный). Сравнение проводить по следующим оценочным элементам: надежность ПС, сопровождаемость, корректность. Критерии оценки (1 или 0)

3. Все сравнение занести в следующую таблицу.

**Практическое занятие №13 «Разработка тестовых пакетов»**

**Цель:** получить навыки разработки тестовых пакетов.

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

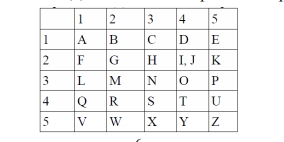
−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

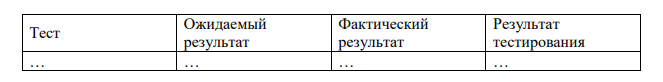
**Теоретические вопросы**

Системные основы разработки требований к сложным комплексам программ. Формализация эталонов требований и характеристик комплекса программ. Формирование требований компонентов и модулей путем декомпозиции функций комплексов программ. Тестирование по принципу «белого ящика».

**Задание № 1.** В Древней Греции (II в. до н.э.) был известен шифр, называемый "квадрат Полибия". Шифровальная таблица представляла собой квадрат с пятью столбцами и пятью строками, которые нумеровались цифрами от 1 до 5. В каждую клетку такого квадрата записывалась одна буква. В результате каждой букве соответствовала пара чисел, и шифрование сводилось к замене буквы парой чисел. Для латинского алфавита квадрат Полибия имеет вид:



Пользуясь изложенным способом создать программу, которая: а) зашифрует введенный текст и сохранит его в файл; б) считает зашифрованный текст из файла и расшифрует данный текст. Задание № 2. Спроектировать тесты по принципу «белого ящика» для программы, разработанной в задании № 1. Выбрать несколько алгоритмов для тестирования и обозначить буквами или цифрами ветви этих алгоритмов. Выписать пути алгоритма, которые должны быть проверены тестами для выбранного метода тестирования. Записать тесты, которые позволят пройти по путям алгоритма. Протестировать разработанную вами программу. Результаты оформить в виде таблиц:



**Задание № 3**. Проверить все виды тестов и сделать выводы об их эффективности..

**Задание № 4.** Оформить отчет.

**Практическое занятие № 14 «Оценка программных средств с помощью метрик»**

**Цель:** Получить навыки по разработке метрик для моделей качества интерфейсов пользователя программных систем

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

**Теоретическая часть**

1. Метрики качества программных систем

Атрибуты (характеристики) программной системы, характеризующие ее качество, измеряются с использованием метрик качества.

Метрика – это комбинация конкретного метода измерения (способа получения значений) атрибута сущности и шкалы измерения (средства, используемого для структурирования получаемых значений).

Метрика определяет (вычисляет) меру атрибута – переменную, которой присваивается значение в результате измерения.

Мера атрибута может быть непосредственной, если она не зависит от мер других атрибутов, либокосвенной, полученной по мерам других атрибутов.

По определению стандарта ISO/IEC 9126-2 метрика качества программной системы представляет собой «модель измерения атрибута, связываемого с характеристикой качества ПС. Служит индикатором одного или многих атрибутов. Ее можно увидеть, например, в левой части большинства уравнений X = A\*B, где X имеет не ту же шкалу, что A или B».

Метрика называется базовой, если в ее основе лежит элементарный метод (примитив) измерения атрибута. По определению того же стандарта «базовая метрика сама по себе не является индикатором характеристики или подхарактеристики качества. Ее можно увидеть, например, в правой части большинства уравнений X =A\*B.  А и В – базовые метрики». То есть базовые метрики используются только в составе модели измерения атрибута.

Для того чтобы правильно пользоваться результатами измерений, для каждой меры нужно идентифицировать шкалу измерения.

Стандарт ISO/IEC 9126-2 рекомендует применять 5 видов шкалы измерения значений (упорядоченных от менее строгих к более строгим):

· номинальная шкала. Это классификационная шкала, выполняющая категоризацию свойств оцениваемого объекта. Категории не упорядочены. Например, дефекты могут классифицироваться на дефекты интерфейса, логики, объявления данных и др. Языки – Fortran, С++, Java и др.;

· порядковая шкала. Позволяет упорядочивать характеристики по возрастанию или убыванию путем сравнения их с базовыми значениями. Например, для уровня серьезности последствий события шкала может включать значения «низкий», «средний», «высокий», «критический». Для уровней СММ – 1, 2, 3, 4, 5. Расстояние между значениями по шкале не играет роли. Характеристики, имеющие номинальную или порядковую шкалу измерения, называются качественными (или категорийными). Все остальные - количественными.

· интервальная шкала. Отмечает существенные различия свойств объекта, «дистанцию» между ними (например, календарные даты или значения плотности дефектов - 1.5 дефекта/KSLOC, 3.5 дефекта/KSLOC и т.д.). Используется в арифметических операциях и операциях сравнения (в данном примере разница равна 2 дефекта/KSLOC). Нулевое значение не допустимо;

· относительная шкала. Значения по этой шкале различаются по отношению к выбранной единице (например, времени, изменяющемся от 0 до бесконечности, или стоимости). Применяя эту шкалу можно рассчитать, например, время между отказами, размер программного компонента в SLOC и др. Считается наиболее предпочтительной шкалой измерений. Позволяет применять широкий спектр инструментов измерения (гистограмм, диаграмм Парето и др.);

· абсолютная шкала. Это специальный случай относительной шкалы. В

этой шкале указывается абсолютное значение величины. Например: «размер программы равен 2К», «число обнаруженных ошибок равно 20».

Измеренное значение метрики само по себе не несет информации об уровне удовлетворения требований к качеству. Для этих целей шкала должна быть разделена на области (ранги), соответствующие различным степеням удовлетворения требований. Примеры деления шкалы:

· деление значений по двум категориям - удовлетворительные и неудовлетворительные значения;

· деление шкалы по четырем категориям.

По мере накопления практики измерений и знаний об измеряемых атрибутах шкалы их измерения могут эволюционировать от менее информативных (номинальной и порядковой) к более информативным (относительной или абсолютной).

1. Познакомьтесь с метриками, которые приведены в примерах теоретической части.

2.  Познакомьтесь с характеристиками и требованиями к ним, которые предъявляются к человеко-машинным интерфейсам программно-технических комплексов а атомной энергетике.

3. Выберите вариант задания

4. Определите метрики для оценки заданной характеристики качества. Для этого выберите метрики из моделей 9126, QUIM и т.л.  или разработайте собственные оригинальные метрики для следующих характеристик пользовательских программных интерфейсов.

5. Задайте метрику в форме спецификации табл. 7

6. Обоснуйте адекватность выбранных метрик.

7. Составьте отчет

Варианты заданий

Таблица 11 Характеристики и требования к ним

|  |  |  |
| --- | --- | --- |
| № | Характеристика | Описание |
| 1. | Безопасность персонала  (Personnel Safety) | Дизайн интерфейса должен обеспечивать минимальную  возможность травм и воздействия вредных материалов |
| 2. | Когнитивная совместимость  (Cognitive Compatibility) | Роль оператора должна состоять из целенаправленных и значимых задач, которые позволяют персоналу поддерживать хорошую осведомленность с АЭС и поддерживать уровень нагрузки, который не настолько высокий, чтобы негативно повлиять на производительность, но достаточной для поддержания бдительности |
| 3. | Физиологическая совместимость  (Physiological Compatibility) | Дизайн интерфейса должен отражать рассмотрениефизиологических характеристик человека, включая визуальное / слуховое восприятие, биомеханику(достижения и движения), характеристики управления, и антропометрии |
| 4. | Простота конструкции  (Simplicity of Design) | ЧМИ должны представлять простой дизайн в соответствии с функциональными требованиями и требованиями    задачи |
| 5. | Согласованность (Consistency) | Должна быть высокая степень согласованности между ЧМИ, процедурами и обучающими системами. В ЧМИ пути системных функций и  деятельности бригады всегда должны быть согласованы, отражать высокую степень стандартизации, и быть в полном соответствии с процедурами и подготовкой кадров |
| 6. | Понимание ситуации  (Situation Awareness) | Информация, представленная пользователям ЧМИ должна быть правильно, быстро и легко понята (например, "непосредственное восприятие"или "определение состояния с одного взгляда" надисплее) и поддерживаться на высоком уровне с целью осведомленности пользователей о статусе системы |
| 7. | Целевая совместимость  (Task Compatibility) | Система должна отвечать требованиям пользователей для выполнения своих задач (в том числе, безопасное завершение работы, осмотр, техническое обслуживание и ремонт).  Данные должны быть представлены в формах и форматах, соответствующих задач (включая, необходимость доступа к подтверждающим данным или необработанным данным в случае отображения более высокого уровня).  Возможность контроля должна охватывать ряд потенциальных действий. Не должно быть ненужной информации или вариантов контроля |
| 8. | Пользовательская модель совместимости  (User Model Compatibility) | Все аспекты системы должны быть совместимы с ментальными (психическими) моделями пользователей(понимание и ожидание поведения системы осуществляется путем подготовки кадров, использования процедур и опыта).Все аспекты системы должны быть совместимы с установленными допущениями, т.е. должны быть выражены в обычной,  привычной, пригодной и функциональной точки зрения, а не абстрактно |
| 9. | Структура  элементов ЧМИ (Organization of HSI Elements) | Структура всех аспектов ЧМИ (от элементов вотдельных дисплеях до отдельных рабочих станций и всейкомнаты управления) должна быть основана на требованияхпользователя и должна отражать общие принципы организации по важности, частоте и порядке использования. Информация критических функций безопасности должна быть доступна всем, работающим в команде,  для обеспечения ее распознавания и сведения  к минимуму поиска данных и ответных мер |
| 10. | Логическая/Явная структура  (Logical/Explicit Structure) | Все аспекты системы (форматы, терминология, последовательность, группировка, и поддержка принятия решений оператора) должна отражать очевидную логику, основанную на требованиях задачи или других непроизвольных обоснованиях. Отношения каждого отображения, управления и обработки данных  для общей задачи/функции должны быть ясными. Структура интерфейса и связанная с ней навигация должны быть сделаны  легкой для пользователей, чтобы было понятно, где они находятся в пространстве данных. Структура интерфейса должна позволить пользователям получить быстрый доступ к данным, не видимым в настоящее время (например, на других страницах дисплей). Ход работы системы и структурированность должны быть ясными для пользователя |
| 11. | Своевременность (Timeliness) | Проектирование системы должны принимать во внимание когнитивные возможности  пользователей, а также связанные с процессом ограничения времени для обеспечения того, чтобы задачи были выполнены в срок. Скорость информационного потока и требования  контроля за исполнением, которые являются слишком быстрыми или слишком медленными могут привести к снижению производительности |
| 12. | Совместимость управления/отображения  (Controls/Displays Compatibility) | Отображения должны быть совместимы с вводимыми данных и требованиями управления |
| 13. | Обратная связь  (Feedback) | Система должна давать полезную информацию о состоянии системы, допустимых операциях, ошибках и восстановлении после ошибки, опасных операциях, и достоверности данных |
| 14. | Когнитивная нагрузка (Cognitive Workload) | Информация, представленная системой должна быстровосприниматься и  пониматься.  Поэтомусистема должна минимизировать требования для вычислений или преобразований в уме и использовать напоминания (ссылаясь на длинные списки кодов, сложные  команды, информацию с одного экрана на другой,или длительные последовательности действий). Исходные данные должны быть обработаны и представлены внепосредственно удобной форме. Исходные данные должны быть доступны для подтверждения |
| 15. | Нагрузка ответа (реакции)  (Response Workload) | Система должна требовать минимальное количество действий для получения результата. Например,одну команду ввода вместо нескольких команд, меню выбора вместо многократных команд, один режим ввода (клавиатура, мышь) вместо смешанного режима.Система не должна требовать ввода избыточных данных,  повторного ввода информации, имеющейся уже в системе,или информации, которую система может генерировать по уже поступившим данным |
| 16. | Гибкость  (Flexibility) | Система должна предоставить пользователю несколько способов для совершения действий ипроверить автоматические действия.  Отображение  и контроль должен быть отформатирован в конфигурации наиболее удобной для задачи.Однако, гибкость должна быть ограничена ситуациями, когда она предлагает преимущества в выполнении задачи (например, для приспособления к различным уровням опыта пользователей) |
| 17. | Руководства и поддержка пользователя (User Guidance and Support) | Система должна обеспечить эффективную "Помощь".Информативные, легкие в использовании рекомендации должны быть предоставлены в ​​он-лайн и офф-лайн режимах, чтобы помочь пользователю понять, как работать с системой |
| 18. | Толерантность  и управление ошибками (Error Tolerance and Control) | Отказоустойчивый дизайн должен предоставляться везде, где сбой может привести к повреждению оборудования, травмам персонала, или непреднамеренной работе критически важного оборудования.Таким образом, система должна вообще быть сконструирована таким образом, чтобы ошибки пользователя не имели серьезных последствий. Надо управлять негативными последствиями ошибок, и сводить их к минимуму. Система должна предлагать простые,понятные уведомления об ошибке, и простые, эффективные методы для восстановления |

**Практическое занятие № 15 «Разработка структуры проекта»**

**Цели:**

На базе лекционного материала по структурам проекта, где рассматривались типовые структуры проекта в общем виде и типовые структуры для проекта «Капитальный ремонт дома», студенты разрабатывают типовые структуры для индивидуального проекта.

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

## *ЗАДАНИЕ*

1. Разработать дерево целей проекта.
2. Разработать структуру продукции проекта.
3. Разработать структуру разбиения работ проекта.
4. Разработать сетевую модель проекта.
5. Разработать организационную структуру проекта.
6. Разработать матрицу ответственности по работам проекта.
7. Разработать структуру затрат проекта.
8. Разработать структуру трудовых ресурсов.
9. Разработать структуру материальных ресурсов.
10. Разработать таблицу основных рисков по проекту и оценить вероятность их наступления. Определить мероприятия по снижению рисков в проекте.

#### МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЗАДАНИЯ:

1. Практическое занятие проводится в форме деловой игры.
2. Учебная группа разбивается на бригады по 3-4 человека, образуя команду проекта.
3. Определяется менеджер проекта.
4. Проводится обсуждение темы в команде.
5. На основе коллегиального мнения разрабатываются структуры проекта по выбранной теме.
6. Разработанные структуры менеджер проекта защищает перед всей группой.

#### Темы проектов:

1. Проект внедрения в компании информационной системы электронного документооборота.
2. Проект создания локальной сети компании.
3. Проект разработки программного продукта.
4. Проект внедрения информационной системы в производственном отделе.
5. Проект создания сайта компании.
6. Проект создания модели бизнес-процессов компании для информационной системы.
7. Проект продвижения на рынок информационных услуг компании.
8. Проект продвижения на рынок нового программного продукта.
9. Проект внедрения информационной библиотечной системы в высшем учебном заведении.
10. Проект организации международной конференции по информационным технологиям.

**Практическое занятие № 16«Разработка модульной структуры проекта (диаграммы модулей)»**

**Целью** : является изучение процесса разработки модульной структуры программного обеспечения, осуществляемого с помощью структурных карт Константайна.

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

**Требования к содержанию, оформлению и порядку выполнения**

Отчет по выполнению лабораторной работы должен содержать: титульный лист, название работы, цель работы и содержательную часть.

В содержательной части отчета по выполнению лабораторной работы для своего варианта требуется привести структурные карты с подробными комментариями к принятым решениям и построенным диаграммам.

**Теоретическая часть**

Теоретические сведения для выполнения лабораторной работы приведены в разделе 3.4 учебно-методического пособия.

**Общая постановка задачи**

Осуществите разработку модульной структуры программного обеспечения задачи, выбранной в первой лабораторной работе, и оформите результат в виде структурной карты, при этом используйте программный продукт EasyCASE Professional Version 4.21.016. Прокомментируйте принятые решения.

**Список индивидуальных данных**

Продолжается работа над задачей, выбранной в первой лабораторной работе.

**Пример выполнения работы**

В соответствии с требованиями, предъявляемыми техническим заданием, и результатами внешнего проектирования (см. предыдущие лабораторные работы) разработаем модульную структуру подсистемы обслуживания клиента по его кредитной карте в банкомате.

В составе программного обеспечения можно выделить следующие программные модули: Головной модуль (Main module), Модуль управления устройством считывания кредитной кары (Credit cart control module), Модуль аутентификации (Autentification module) и Модуль получения и обработки запроса на обслуживание (Reception and processing module). Кроме этого в состав ПО необходимо включить модуль данных кредитной карты (Credit cart data).

Основной функцией Головного модуля является организация общего управления поведением подсистемы и выполняет вызов всех остальных программных модулей.

Модуль управления устройством считывания кредитной карты выполняет функции связанные с обработкой кредитной карты: ввод, считывание хранящейся на ней информации, удаление.

Модуль аутентификации выдает сообщение клиенту на ввод ключевых данных, выполняет получение пароля и проверку его правильности.

Модуль получения и обработки запроса на обслуживание выполняет следующие функции: Получение запроса на обслуживание и проверка возможности его исполнения, Обработка запроса на обслуживание, включающая такие действия как:

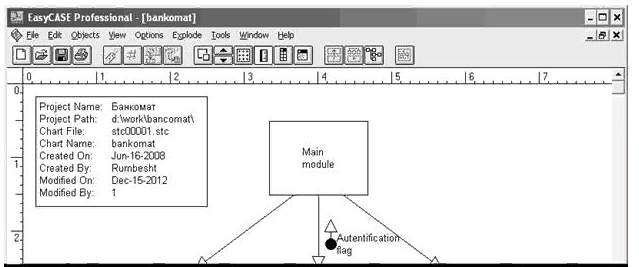
· обработка внутренней банковской документации по клиенту;

· распечатка баланса клиента;

· выдача наличных денег и информирование компьютера банка об изъятых из банка деньгах;

· распечатка операции клиента.

На рис. Л5.1 приведена структурная карта, демонстрирующая отношения между указанными модулями системы.



Согласно этой диаграмме головной модуль обращается к модулям управления устройством считывания кредитной карты, аутентификации и получения и обработки запроса на обслуживание. Вызов указанных модулей осуществляется согласно внутренней логики головного модуля, реализующей следующий сценарий: При инициации действий со стороны клиента головной модуль вызывает модуль управления устройством считывания кредитной карты для ее ввода и считывания с нее информации. После завершения считывания управление возвращается головному модулю, который затем обращается к модулю аутентификации. Модуль аутентификации проверяет подлинность клиента и вместе с результатом этой проверки возвращает управление головному модулю. В зависимости от результатов аутентификации головной модуль либо вызывает модуль управления устройством считывания для удаления кредитной карты, либо обращается к модулю получения и обработки запроса на обслуживание для предоставления требуемого сервиса. Если осуществляется вызов получения и обработки запроса на обслуживание, то после завершения его работы головной модуль обращается к модулю управления устройством считывания для удаления кредитной карты.

Обмен данными между программными модулями осуществляется через общую область памяти, в которую модуль управления устройством считывания помещает данные о пароле (Parol), атрибуты клиента (Client Attributes) и лимит денег на счету (Limit of money). Модуль аутентификации получает из этой общей области памяти сведения о пароле и возвращает в головной модуль управляющий параметр Autentification flag, содержащий результат аутентификации. Модуль получения и обработки запроса на обслуживание для своей работы получает из общей области памяти атрибуты клиента и лимит денег на счету.

**Контрольные вопросы к защите**

1. Цель разработки модульной структуры.

2. Понятие программного модуля, передачи управления, организации связи по управлению и по данным.

3. Виды связности модулей.

4. Виды целостности модулей.

5. Типовые модульные структуры.

6. Проектирование модульной структуры с помощью структурных карт.

**Практическое занятие №17 «Разработка перечня артефактов и протоколов проекта»**

**Цель работы**: ознакомление с процедурой разработки эскизного проекта на программный продукт, с применением ГОСТ 19.105 -78 «Пояснительная записка к техническому проекту» и ГОСТ 19.404 – 79 «Пояснительная записка. Требования к содержанию и оформлению». Конкретное содержание работ на стадии эскизного проекта и их объем определяет степень сложности разрабатываемого ПП.

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

Результатом выполнения данной стадии является полное описание архитектуры ПП. Как правило, это описание делается на нескольких уровнях иерархии. На верхнем уровне детализации выделяются основные подсистемы, которым присваиваются имена, устанавливаются связи между подсистемами, их функции, получаемые путем декомпозиции предполагаемых функций ПП. Затем процедура декомпозиции выполняется для каждой подсистемы, выделяются модули, составляющие данную подсистему. В конечном итоге, получается иерархически организованная система, состоящая из уровней, каждый из которых представляет собой совокупность взаимосвязанных модулей. В качестве примера ниже приводится фрагмент расширенного описания работ стадии эскизного проекта:

• разработка плана совместных работ на разработку ПП;

• разработка и обоснование математической модели системы и описание результатов моделирования;

• разработка и обоснование алгоритмов и временных графиков функционирования ПП по всем режимам работы;

• разработка и обоснование ресурсов памяти для реализации алгоритмов;

• разработка перечня документов на ПП;

• разработка и обоснование структуры БД, внешних входных и выходных данных;

• разработка и обоснование алгоритмов информационного обеспечения;

• разработка и обоснование набора тестов для проверки ПП;

• разработка и обоснования организации работ по развитию ПП;

Результатом выполнения данной работы является эскизный проект, оформленный в соответствии с ОС ТУСУР.

**Практическое занятие №18 «Настройка работы системы контроля версий (типов импортируемых файлов, путей, фильтров и др. параметров импорта в репозиторий)»**

**Цель работы**

1. Изучить на практике понятия и компоненты систем контроля версий (СКВ), приемы работы с ними.

1. Освоить специализированное ПО и распространенный сервис для работы с распределенной СКВ Git — TortoiseGit и GitHub.com.

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

**Общие указания к выполнению лабораторной работы**

1. Система контроля версий Git представляет собой набор программ командной строки. Доступ к ним можно получить из терминала (в \*nix) или из специальной консольной оболочки Git Bash (в Windows). Однако, лабораторная работа ориентирована на применение графической надстройки TortoiseGit (аналог в Linux — RabbitVCS). TortoiseGit работает не как отдельная программа, а встраивается в контекстные меню «Проводника» Windows. Вместе с Git для Windows поставляется также программа gitk (Git GUI) — она гораздо менее популярна и пользоваться ей для ЛР не следует.

2. Задания необходимо читать внимательно и полностью. Благодаря тому, что Git является распределенной СКВ, резервную копию локального хранилища можно сделать простым копированием или архивацией.

1. В отчет можно включать снимки экрана, сообщения и комментарии по своему усмотрению с тем, чтобы было удобно пояснять сделанное, опираясь на отчет.

Задание на лабораторную работу ,

1. Отработать навыки использования хранилища на локальной машине. 1.1. Настроить Git, указав имя и e-mail разработчика для подписи commit-ов. Указание. Диалог настроек вызывается пунктом TorotiseGit → Settings контекстного меню любого каталога, нужная вкладка называется «Git». Задавать следует глобальные настройки (для всех хранилищ), установив переключатель «Global».

1.2. Создать хранилище для учебного проекта.

1.3. Совершить несколько commit-ов.

1.3.1. Скопировать sdt.h в каталог хранилища и создать файл main.cpp со включением sdt.h и пустой функцией main().

1.3.2. Добавить в программу ввод двух целых чисел с приглашением. Указание № 1. После выполнения каждого подпункта необходимо убеждаться, что программа работает, и совершать commit изменений.

Указание № 2. Следите за тем, какие файлы отмечены в списке для commitа изменений в них, — кроме main.cpp и, иногда, sdt.h, больше никаких других не нужно.

1.4. Предотвратить автоматическое добавление в хранилище файлов, не нуждающихся в контроле версий, — \*.o и \*.exe. Правило об игнорировании следует помещать в файл .gitignore в корневом каталоге хранилища («.gitignore in repository root»). Этот файл также попадает под контроль версий, поэтому после создания правил требуется совершить commit изменений в файле .gitignore.

1.5. Добавить в программу вывод суммы введенных чисел и совершить commit.

1.6. Просмотреть историю (журнал) хранилища.

1.7. Просмотреть разность (diff) между пунктами истории 1.3.2 и 1.5. 2. Освоить передачу истории хранилища по сети.

2.1. Организовать общее хранилище на удаленном сервере.

2.1.1. Зарегистрироваться на GitHub.

2.1.2. Создать пустое удаленное хранилище с любым наименованием

Указание. Вопреки инструкции на GitHub, добавлять в хранилище файл README.md не нужно, удаленное хранилище должно быть пустым.

2.1.3. Разрешить пользователям-преподавателям совершать commit-ы. Требуется на странице хранилища выбрать «Settings» (справа), далее «Collaborators», где ввести имена пользователей, которым будет предоставлен полный доступ к хранилищу (эти имена можно узнать у лаборантов).

2.1.4. Настроить локальное хранилище для синхронизации с удаленным. Необходимо в контекстном меню каталога локального хранилища выбрать TortoiseGit → Settings, где перейти к пункту Remote . Достаточно ввести условное имя удаленного хранилища «origin» и его адрес, который отображается на webстранице удаленного хранилища в разделе «Quick setup» (вариант HTTP). От загрузки сведений о ветвлениях в удаленном хранилище отказаться.

2.2. Передать локальное хранилище на удаленный сервер (push). Замечание. Здесь и далее при взаимодействии с удаленным сервером потребуется вводить имя пользователя и пароль, с которыми выполнялась регистрация на GitHub.

Перейти к странице хранилища на GitHub (обновить её) и ознакомиться с возможностями просмотра содержимого через web-интерфейс.

2.4. Загрузить копию удаленного хранилища на локальную машину (clone). Замечание. Целью является имитация совместной работы с удаленным хранилищем. Для этого на одной машине организуются 2 локальных хранилища: созданное в пункте 1.1 (RepoA) и загруженное с удаленного сервера (RepoB). Указание. Диалог «Git clone» следует вызывать из контекстного меню каталога вне локального хранилища. В качестве URL потребуется указать адрес удаленного хранилища, а в качестве Directory — имя каталога для нового локального хранилища.

2.5. Сымитировать параллельную работу над проектом.

2.5.1. В локальном хранилище RepoB добавить в программу печать разности введенных чисел, сделать commit и передать изменения на сервер. 2.5.2. В локальном хранилище RepoA добавить над функцией main() комментарий о том, что программа является учебной, сделать commit, но не отправлять изменений на сервер.

2.6. На странице хранилища на GitHub перейти в раздел Commits и ознакомиться с возможностью просмотра истории изменений через web-интерфейс.

2.7. В локальном хранилище RepoA выполнить загрузку с сервера новейших ветвлений и изменений (fetch) и просмотреть журнал хранилища.

**Практическое занятие № 19 «Разработка и интеграция модулей проекта (командная работа)»**

**Цель работы** состоит в приобретении навыков работы в составе бригады при

разработке программного продукта

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

**Краткие теоретические и учебно-методические материалы по теме** практического занятия

В настоящее время сложность промышленных приложений и систем такова, что процесс их разработки стал практически неуправляемым. Кроме того, их развертывание на сотнях компьютеров, расположенных в разных местах, значительно раздвигает границы процесса разработки.

Один человек не способен создать приложение масштаба предприятия. Ни один разработчик просто не удержит в голове все требования к системе и варианты проекта. Поэтому сегодня разработкой промышленных систем занимаются проектные группы, и все обязанности распределяются среди членов группы. Существует две основные модели организации коллектива при разработке ПО:

1) иерархическая модель;

2) модель группы.

Работа в коллективе отличается определенными сложностями. Иерархическая модель организации, определяет начальников и подчиненных. Однако если в современных производственных средах один менеджер проекта отвечает за все тонкости разработки и принимает все важные решения, возникает множество проблем, ведущих к провалу проекта. Опыта одного человека чаще всего недостаточно для быстрого решения задачи и для интеграции приложения в существующую инфраструктуру. В организациях, построенных на основе иерархической модели, затруднен обмен информацией - в этой модели он, по определению, осуществляется через посредников. Вся информация иерархических групп «фильтруется» тремя или четырьмя менеджерами, что значительно повышает вероятность утери самого важного. Часто такое отсеивание идей происходит при прохождении сообщения от разработчика, непосредственно занимающегося проектом, к высшему руководству. Естественно, некоторые участники «выпадают» из процесса, что снижает эффективность их труда и повышает вероятность провала проекта.

Чтобы сгладить недостатки иерархической модели, в проектной группе предусматривается распределение обязанностей руководителя между ленами коллектива.

При этом за проект отвечает не один человек, а все члены группы - каждый за свой участок.

**Задания для практического занятия:**

1.Распределение ролей в бригаде (см. приложение).

2.Выполнить работу в соответствии с заданием преподавателя.

Вопросы для закрепления теоретического материала к практическому занятию:

Основные модели организации коллектива при разработке ПО:

Недостатки коллективного подхода.

Обязанности членов группы.

Модель проектной группы. Цели и роли

Задачи проектной группы.

Порядок выполнения отчета по практической работе

Защита отчета по практической работе заключается в предъявлении

преподавателю полученных результатов (на экране монитора и печатном виде),

демонстрации полученных навыков и ответах на вопросы преподавателя.

2 Ответить на контрольные вопросы.

**Практическое занятие №20 «Отладка отдельных модулей программного проекта»**

**Цель.** Изучить процесс отладки программного обеспечения ручным методом.

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

**Ход работы**

1. Ознакомиться с теоретической частью.
2. Выполнить практическое задание.
3. Ответить на контрольные вопросы.
4. Оформить отчет.

**Теоретическая часть**

*Отладка* – это процесс локализации и исправления ошибок, обнаруженных при тестировании программного обеспечения.

*Локализацией* называют процесс определения оператора программы, выполнение которого вызвало нарушение нормального вычислительного процесса. Для исправления ошибки необходимо определить ее причину, т.е. определить оператор или фрагмент, содержащие ошибку. Причины ошибок могут быть как очевидны, так и очень глубоко скрыты. В целом сложность отладки обусловлена следующими причинами:

* требует от программиста глубоких знаний специфики управления используемыми техническими средствами, операционной системы, среды и языка программирования, реализуемых процессов, природы и специфики различных ошибок, методик отладки и соответствующих программных средств;
* психологически дискомфортна, так как необходимо искать собственные ошибки и, как правило, в условиях ограниченного времени;
* возможно взаимовлияние ошибок в разных частях программы, например, за счет затирания области памяти одного модуля другим из-за ошибок адресации;
* отсутствуют четко сформулированные методики отладки.

**Классификация ошибок**

В соответствии с этапом обработки, на котором появляются ошибки, различают:

* *синтаксические ошибки* – ошибки, фиксируемые компилятором (транслятором, интерпретатором) при выполнении синтаксического и частично семантического анализа программы;
* *ошибки компоновки* – ошибки, обнаруженные компоновщиком (редактором связей) при объединении модулей программы;
* *ошибки выполнения* – ошибки, обнаруженные операционной системой, аппаратными средствами или пользователем при выполнении программы.

**Методы отладки программного обеспечения**

Отладка программы в любом случае предполагает обдумывание и логическое осмысление всей имеющейся информации об ошибке. Большинство ошибок можно обнаружить по косвенным признакам посредством тщательного анализа текстов программ и результатов тестирования без получения дополнительной информации. При этом используют различные методы:

* ручного тестирования;
* индукции;
* дедукции;
* обратного прослеживания.

**Метод ручного тестирования**

Это – самый простой и естественный способ данной группы. При обнаружении ошибки необходимо выполнить тестируемую программу вручную, используя тестовый набор, при работе с которыми была обнаружена ошибка. Метод очень эффективен, но не применим для больших программ, программ со сложными вычислениями и в тех случаях, когда ошибка связана с неверным представлением программиста о выполнении некоторых операций. Данный метод часто используют как составную часть других методов отладки.

Общая методика отладки программных продуктов, написанных для выполнения в операционных системах MS DOS и Win32:

*1 этап –*изучение проявления ошибки;

*2 этап –*определение локализации ошибки;

*3 этап –*определение причины ошибки;

*4 этап –*исправление ошибки;

*5 этап –*повторное тестирование.

Процесс отладки можно существенно упростить, если следовать основным рекомендациям структурного подхода к программированию:

* программу наращивать «сверху-вниз», от интерфейса к обрабатывающим подпрограммам, тестируя ее по ходу добавления подпрограмм;
* выводить пользователю вводимые им данные для контроля и проверять их на допустимость сразу после ввода;
* предусматривать вывод основных данных во всех узловых точках алгоритма (ветвлениях, вызовах подпрограмм).

*Спецификация программы*, программная спецификация (program specification) - точная и полная формулировка определенной задачи или группы задач, содержащая сведения, необходимые для построения алгоритма их решения. Содержит описание результата, который должен быть достигнут с помощью конкретной программы, а также действий, выполняемых программой для достижения конечного результата без упоминания того, как указанный результат достигается

**Практическая часть**

1. Запишите вариант в отчет.
2. Согласно поставленной задаче выполните ручную отладку:

* Опишите математическую модель задачи с указанием имен и назначения переменных;
* Опишите спецификацию программы;
* Запишите алгоритм программы;
* Выполните отладку логики программы методом «грубой силы» с помощью соседа;
* Составьте тестовые наборы для проверки функционала системы.

1. Результаты выполнения практического задания запишите в отчет.

**Контрольные вопросы**

1. В чем заключается ручная отладка ПО?
2. На каком этапе проводится ручная отладка?
3. Опишите методы отладки.

**Практическое занятие №21 «Организация обработки исключений»**

**Цель работы**: знакомство с механизмом обработки исключительных ситуаций.

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

Форма проведения: выполнение индивидуального задания. Рекомендации по подготовке к лабораторной работе: при подготовке к лабораторной работе необходимо изучить теоретический материал по теме — учебное пособие [4], стр. 163 — 165, учебное пособие [5], стр. 271 — 274. При изучении материала особое внимание уделите следующим вопросам: понятие исключительной ситуации, синтаксис обработки исключительной ситуации, этапы обработки исключительных ситуаций. Порядок проведения занятия

1. Самостоятельно определить 2 — 3 исключительные ситуации, которые могут возникнуть при работе с объектами ранее разработанных классов.

2. Реализовать программно механизм обработки исключительных ситуаций.

3. Отладить и протестировать программу.

4. Защитить работу.

**Практическое занятие № 22«Отладка проекта»**

**Цель работы**: знакомство с механизмом отладки проектов.

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

Данное практическое занятие выполняется частями в составе двух предыдущих практических занятий. Часть 1 выполняется в составе ["Инсталляция рабочего места архитектора проекта, основные функции и возможности, создание архитектуры проекта согласно техническому заданию на проект"](https://www.intuit.ru/studies/courses/499/355/lecture/8465)(3 часа). Части 2 и 3 выполняются в составе ["Инсталляция рабочего места разработчика проекта, разработка фрагментов проекта"](https://www.intuit.ru/studies/courses/499/355/lecture/8468)работы .

Тестирование (*testing*) —процесс выполнения программы (или части программы) с намерением (или целью) найти ошибки.

Комплексное тестирование (*system testing*) — *контроль* и/или *испытание* системы по отношению к исходным целям. Комплексное тестирование является *процессом контроля*, если оно выполняется в моделируемой среде, и процессом испытания, если выполняется в среде реальной, жизненной.

*Отладка* (*debugging*) не является разновидностью тестирования. Хотя слова "*отладка*" и "тестирование" часто используются как синонимы, под ними подразумеваются разные виды деятельности. Тестирование — *деятельность*, направленная на обнаружение ошибок; *отладка* направлена на установление точной природы известной ошибки, а затем — на исправление этой ошибки. Эти два вида деятельности связаны — результаты тестирования являются исходными данными для отладки.

Нельзя сказать, что комплексное тестирование является независимым этапом реализации IT проекта. Скорее, комплексное тестирование – ряд распределенных во времени мер, направленных на выявление соответствия реализуемого решения исходным требованиям и ограничениям.

В данной лабораторной работе мы рассмотрим три основных, с нашей точки зрения, элемента комплексного тестирования.

#### Комплексное тестирование в VSTS 2008.

По сути, реальное комплексное тестирование можно провести лишь в реальной производственной среде, для которой создается приложение. Однако, если подойти к комплексному тестированию как к процессу контроля (см. определение выше), то можно реализовать его часть средствами VSTS 2008.

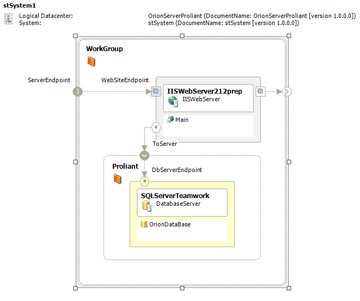
Основным инструментом комплексного тестирования будет конструктор развертывания. Он позволяет выполнить, так называемый, тест на развертывание, в ходе которого проверяется возможность эксплуатации спроектированного продукта в конкретном логическом центре данных. Этот тест завершается успехом или неудачей (если параметры и ограничения приложений и центра данных оказываются несовместимыми), в последнем случае выводится информация, необходимая в устранении проблем.

Создание *диаграммы развертывания* описано в ["Инсталляция рабочего места архитектора проекта, основные функции и возможности, создание архитектуры проекта согласно техническому заданию на проект"](https://www.intuit.ru/studies/courses/499/355/lecture/8465).

##### Часть 1. Тест на развертывание (3 часа)

Тест на развертывание должен выполняться в составе ["Инсталляция рабочего места архитектора проекта, основные функции и возможности, создание архитектуры проекта согласно техническому заданию на проект"](https://www.intuit.ru/studies/courses/499/355/lecture/8465)(освоение рабочего места архитектора). Цель теста – выявить возможность реализации проекта с уже определенной архитектурой в заданном логическом центре данных.

Диаграмма развертывания:

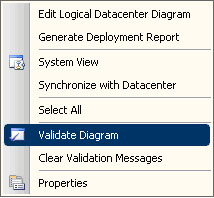


[увеличить изображение](https://www.intuit.ru/EDI/18_03_15_2/1426630694-20867/tutorial/522/objects/19/files/07-01.jpg)  
**Рис. 19.1.**

Большая часть проверок на возможность развертывания будет проводиться по мере формирования схемы. Например, если вы попытаетесь разместить приложение, для которого требуется обеспечить взаимодействие по протоколу HTTP, на хосте, не имеющем такой конечной точке, конструктор не позволит это сделать.

Более сложные нарушения ограничений могут быть неочевидными, пока конструктор не выполнит анализ схемы развертывания.

Для того, чтобы выполнит такой анализ, щелкните правой кнопкой мыши диаграмму развертывания и выберите команду Validate Diagram.



**Рис. 19.2.**

В процессе анализа просматриваются все документы, связанные с процедурой развертывания, и выводятся сообщения об ошибках и предупреждениях. Их отсутствие означает, что тест пройден успешно.

###### **Задание:**

В ходе выполнения ["Инсталляция рабочего места архитектора проекта, основные функции и возможности, создание архитектуры проекта согласно техническому заданию на проект"](https://www.intuit.ru/studies/courses/499/355/lecture/8465)провести тест на развертывание, составить отчет о его результатах и сделать вывод о соответствии логического *цента* данных (среды разработки) требованиям архитектуры реализуемого проекта. В случае выявления ошибок в процессе выполнения теста – исправить ошибки в проектировании архитектуры проекта.

##### Часть 2. Отладка проекта (1,5 часа)

Отладка проекта является процессом, длящимся на протяжении всего времени работы над проектом. Ошибки будет выявляться всегда, а соответственно и отладка должна проводиться регулярно

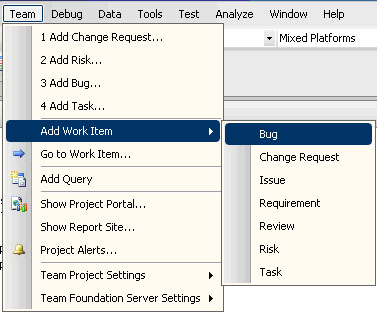
Данная часть задания должна выполняться в составе ["Инсталляция рабочего места разработчика проекта, разработка фрагментов проекта"](https://www.intuit.ru/studies/courses/499/355/lecture/8468)

Поскольку отладка подразумевает исправление ранее выявленных ошибок, то средствами VSTS этот процесс может быть налажен следующим образом:

1. В процессе выполнения какого-либо теста (предполагается, что чаще всего ошибки находит тестировщик) выявляется ошибка.
2. Создается рабочий элемент "Ошибка" (*Bug*)
3. Создается рабочий элемент "Задача" (Task), в котором разработчику указывается информация об ошибке и указание исправить ее.

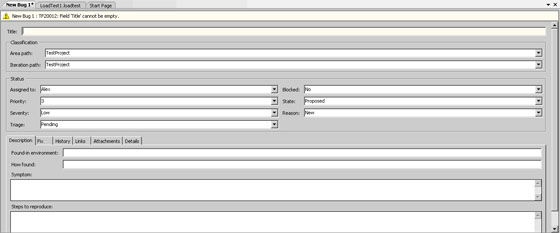
При выявлении ошибок в ходе выполнения ["Инсталляция рабочего места разработчика проекта, разработка фрагментов проекта"](https://www.intuit.ru/studies/courses/499/355/lecture/8468)часть из них должна оформляться в идее рабочего элемента "Ошибка", где должно присутствовать подробное описание действий приведших к сбою, также, как правило, ошибке присваивается определенный номер, в соответствии с принятыми стандартами разработки. Затем должен создаваться рабочий элемент "Задача" с указанием исправить ошибку № \*\*\*\*, задача назначается одному – двум исполнителям, которые должны исправить ошибку. По исправлении, исполнители документируют причину ошибки и внесенные изменения и "закрывают" соответствующие рабочие элементы "ошибка" и "задача".

Процесс создания рабочего элемента "Ошибка".



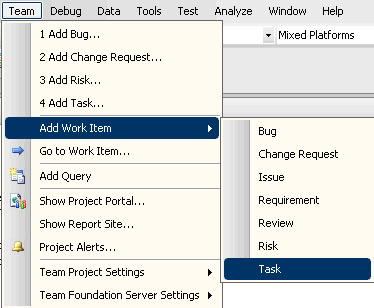
**Рис. 19.3.**

В окне создания рабочего элемента указывается его название, приоритет, итерация, на которой выявлена ошибка и подробное описание действий, приведших к возникновению ошибки.



[увеличить изображение](https://www.intuit.ru/EDI/18_03_15_2/1426630694-20867/tutorial/522/objects/19/files/07-04.jpg)  
**Рис. 19.4.**

Создание рабочего элемента "Задача".



**Рис. 19.5.**

Процесс создания рабочего элемента "Задача" мало отличается от создания рабочего элемента "Ошибка". Разница в том, что указывается не последовательность действий, а задачи, требующие решения.

В сущности, после создания элемента "Задача", отладкой начинает заниматься разработчик, а именно исправлением какой-то конкретной ошибки, после чего вновь проводится тестирование.

###### **Задание:**

Провести отладку выявленных ошибок, процесс отладки см. выше.

##### Часть 3. Сборка проекта (1,5 часа)

Данная часть выполняется в составе ["Инсталляция рабочего места разработчика проекта, разработка фрагментов проекта"](https://www.intuit.ru/studies/courses/499/355/lecture/8468).

Также к комплексному тестированию можно отнести сборку проекта, поскольку при сборке в соответствии со сценарием компилируется весь имеющийся код, и выполняются определенные тесты.

Подробнее процесс создания сборки описан в ["Инсталляция рабочего места разработчика проекта, разработка фрагментов проекта"](https://www.intuit.ru/studies/courses/499/355/lecture/8468).

###### **Задание**

Создать сценарий сборки и выполнить ее. Обязательно назначить 1 - 2 теста к выполнению в процессе сборки, исключая нагрузочные тесты (см. ["Инсталляция рабочего места разработчика проекта, разработка фрагментов проекта"](https://www.intuit.ru/studies/courses/499/355/lecture/8468))

**Практическое занятие № 23 «Инспекция кода модулей проекта»**

**Цель работы**: Приобретение навыков инспекции кода программных модулей.

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

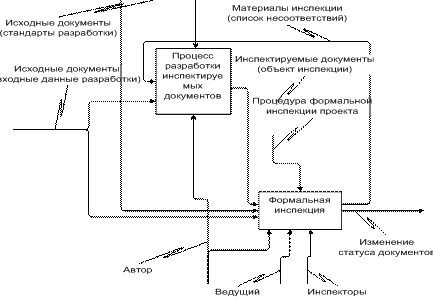
Не во всех случаях возможна разработка автоматических или хотя бы четко формализованных ручных тестов для проверки функциональности программной системы. В некоторых случаях выполнение программного кода, подвергаемого тестированию, невозможно в условиях, создаваемых тестовым окружением (например, во встроенных системах, если программный код предназначен для обработки исключительных ситуаций, создаваемых только при установке системы на реальное оборудование). В других случаях верифицируется не программный код, а проектная документация на систему, которую нельзя "выполнить" или создать для нее отдельные тестовые примеры. И в тех и в других случаях обычно прибегают к методу экспертных исследований программного кода или документации на *корректность* или *непротиворечивость*.

Такие экспертные исследования обычно называют инспекциями или просмотрами. Существует два типа инспекций – неформальные и формальные.

Формальная инспекция является четко управляемым процессом, структура которого обычно четко определяется соответствующим стандартом проекта. Таким образом, все формальные инспекции имеют одинаковую структуру и одинаковые выходные документы, которые затем используются при разработке.

Факт начала формальной инспекции четко фиксируется в общей базе данных проекта. Также фиксируются документы, подвергаемые инспекции, списки замечаний, отслеживаются внесенные по замечаниям изменения. Этим формальная инспекция похожа на автоматизированное тестирование – списки замечаний имеют много общего с отчетами о выполнении тестовых примеров.

В ходе формальной инспекции группой специалистов осуществляется независимая проверка соответствия инспектируемых документов исходным документам. Независимость проверки обеспечивается тем, что она осуществляется инспекторами, не участвовавшими в разработке инспектируемого документа. Входами процесса формальной инспекции являются инспектируемые документы и исходные документы, а выходами – материалы инспекции, включающие *список* обнаруженных несоответствий и решение об изменении статуса инспектируемых документов. [рис. 17.1](https://www.intuit.ru/studies/courses/1040/209/lecture/5407?page=1#image.17.1) иллюстрирует *место* формальной инспекции в процессе разработки программных систем.



**Рис. 17.1.**Место формальной инспекции в процессе разработки программных систем

### Этапы формальной инспекции и роли ее участников

Процесс формальной инспекции состоит из пяти фаз:

* инициализация;
* планирование;
* подготовка (экспертиза);
* обсуждение;
* завершение.

#### Инициализация

Руководитель проекта (преподаватель) выбирает объект инспекции (код, написанный на КР). Затем он назначает участников формальной инспекции: авторов (студенты, написавшие код: 2 или 3 человека), ведущего (сам преподаватель) и нескольких инспекторов (студенты, не участвовавших в написании инспектируемого кода: 4 или 5 человек). Ведущий также выполняет роль инспектора; остальные участники выполняют только одну роль.

Эта фаза формальной инспекции проводится на семинаре 6 с использованием раздаточного материала.

#### Планирование

Считаем, что инспектируемые документы размещены в базе данных проекта, а их статус соответствует готовности к формальной инспекции. Считаем, что статус этих документов изменен так, чтобы отметить начало формальной инспекции, и менять их нельзя.

После этого ведущий заносит в бланк инспекции идентификаторы инспектируемых и исходных документов и номера их версий, список участников с указанием их ролей и дату фактического начала процесса инспекции, т.е. того момента, когда инспектируемые документы были переведены в состояние Review.

Подготовив бланк инспекции и определив время и место собрания, ведущий должен известить участников инспекции о времени и месте и разослать им подготовленный бланк инспекции.

Эта фаза формальной инспекции проводится на ["Тестирование программного кода (покрытия)"](https://www.intuit.ru/studies/courses/1040/209/lecture/5390)(раздается раздаточный материал).

#### Подготовка

Получив назначение с прикрепленным к нему бланком инспекции, исходные и инспектируемые документы, инспекторы детально изучают инспектируемые документы, руководствуясь списком контрольных вопросов.

Перед началом просмотра исходного кода рекомендуется отметить пункты требований, на соответствие которым проверяется исходный код, а также записать обоснования того, почему эти требования не могут быть проверены в автоматическом режиме. После этого можно переходить к просмотру собственно исходного кода. Все пометки, которые придется вносить в ходе инспектирования в исходный код, необходимо делать не в файле, который будет выдан инспекторам, а в его копии, которая потом будет подшита к материалам инспекции. Копия может быть в том же формате, что и исходный файл, либо распечатана на бумаге или выведена в формат DOC, PDF или аналогичный, допускающий комментирование.

Рекомендуется делать пометки, поясняющие, почему именно данный участок кода реализует требования. Такие пометки помогут на этапе собрания.

Рекомендуется также проверять наличие участков, гарантирующих робастность, даже если требования прямо не определяют необходимости обработки недопустимых значений. В случае, если потенциально возможна некорректная работа программы из-за отсутствия обработчиков неверных значений, рекомендуется отметить это в списке замечаний.

Все обнаруженные несоответствия должны быть точно локализованы, сформулированы и записаны.

Эта фаза формальной инспекции проходит в виде домашнего задания семинара 6 студентам, которые назначены инспекторами.

**Обсуждение**

Обсуждение проводится в форме одного собрания.

В ходе обсуждения ведущий синхронизирует работу участников, перечисляя номера или идентификаторы требований (в нашем случае в ходе инспекции проверяется соответствие исходного кода требованиям). По мере продвижения по документу инспекторы прерывают ведущего в тех местах, к которым у них имеются замечания. В случае отсутствия разногласий ведущий фиксирует несоответствие и продолжает продвижение по документу.

Если мнения участников по высказанному замечанию расходятся, то ведущий управляет дискуссией, последовательно предоставляя слово всем желающим высказаться, причем автор пользуется правом внеочередного предоставления слова. Если в результате дискуссии изменилась формулировка замечания, то ведущий записывает эту новую формулировку, затем зачитывает ее и, если все участники с ней согласны, продолжает продвижение по документу.

Результатом дискуссии может также быть признание отсутствия проблемы. В этом случае ведущий убеждается в том, что все с этим согласны, и продолжает продвижение по документу.

Участники должны стремиться обозначить проблемы, но не искать их решения. Достижение консенсуса по спорным вопросам также не является целью дискуссии. Если имеется расхождение во мнениях, то должны быть зафиксированы все альтернативные мнения. Ведущий должен прервать дискуссию, если оценивает ее как непродуктивную.

Все участники обязаны уважительно относиться к оппонентам, не перебивать говорящего и высказываться тогда, когда ведущий предоставит им слово. Не допускаются параллельные обсуждения узким составом – каждый участник обязан адресовать свои высказывания всему собранию, а не соседу.

Необходимо также избегать критики и оценки квалификации коллег. Целью инспекции является повышение качества инспектируемых документов, а не оценка квалификации автора или других участников инспекции.

В ходе обсуждения необходимо в бланке инспекции проставить ответы на контрольные вопросы и зафиксировать замечания. Для этого ведущий последовательно зачитывает контрольные вопросы. При отсутствии у всех инспекторов замечаний, нарушающих сформулированное в вопросе свойство, против вопроса ставится отметка (галочка) в графе "Да"; в противном случае отметка ставится в графе "Нет", а в графе "Ссылка на несоответствие" перечисляются номера соответствующих несоответствий, записанных в таблице для несоответствий, которая помещена в конце бланка инспекции. Отметка в графе "Неприменимо" ставится только в том случае, когда сформулированное в соответствующем вопросе свойство не может быть оценено для данного объекта инспекции; в этом случае в графе "Ссылка на несоответствие" записывается обоснование невозможности оценить данное свойство.

Для облегчения труда автора инспектируемого документа по исправлению замечаний каждое замечание, признанное на собрании существенным, рекомендуется точно трассировать на строки исходного кода и требований.

В конце обсуждения участники принимают решение о возможности принятия объекта инспекции в имеющейся версии либо о необходимости внесения исправлений и проведения повторной инспекции в полной или сокращенной форме. Объект инспекции может быть принят в имеющейся версии только при отсутствии несоответствий. Решение о проведении повторной инспекции в сокращенной форме принимается только в том случае, если все участники с этим согласны. Если хотя бы один из участников настаивает на полной форме повторной инспекции, то повторная инспекция должна проводиться в полной форме. Мнение ведущего учитывается наравне с мнениями других участников. Принятое решение фиксируется ведущим на бланке инспекции и заверяется подписями всех участников.

Теоретически возможна ситуация, когда автор не согласен ни с одним из зафиксированных замечаний, а инспекторы настаивают, что несоответствия существуют. В таком случае невозможно принять решение об изменении статуса инспектируемых документов, поэтому инспекция должна быть отложена, а решение проблемы вынесено за рамки процесса формальной инспекции.

На бланке инспекции также фиксируется продолжительность собрания и время, затраченное каждым из участников на подготовку.

Эта фаза формальной инспекции проходит на текущем семинаре.

**Завершение**

В конце собрания, по окончании обсуждения, инспекторы сдают ведущему свои рабочие материалы, которые включают в себя распечатки инспектируемых документов с пометками и бланки инспекции. Ведущий складывает эти материалы в прозрачную папку вместе с экземпляром бланка инспекции, заполненным в ходе обсуждения, причем титульный лист бланка инспекции должен лежать сверху, чтобы можно было по нему идентифицировать папки.

После собрания ведущий изменяет статус инспектируемых документов в соответствии с принятым решением – либо им присваивается статус "Принят", либо "Переработать".

В последнем случае необходима повторная инспекция, вид которой уточняется кратким комментарием (не проводится).

Эта фаза формальной инспекции проходит на текущем семинаре.

**Практическое занятие № 24 «Тестирование интерфейса пользователя средствами инструментальной среды разработки»**

**Цель работы** Познакомиться с основными элементами управления (виджетами) и приобрести навыки проектирования графического интерфейса пользователя.

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

1.1. Теоретические сведения Графический интерфейс пользователя (GUI) — разновидность пользовательского интерфейса, в котором элементы интерфейса (меню, кнопки, значки, списки и т. п.), представленные пользователю на дисплее, исполнены в виде графических изображений. В GUI пользователь имеет произвольный доступ (с помощью устройств ввода — клавиатуры, мыши, джойстика и т. п.) ко всем видимым экранным объектам (элементам интерфейса) и осуществляет непосредственное манипулирование ими. Графический интерфейс пользователя является частью пользовательского интерфейса и определяет взаимодействие с пользователем на уровне визуализированной информации. Можно выделить следующие виды графического интерфейса пользователя: − простой: типовые экранные формы и стандартные элементы интерфейса, обеспечиваемые самой подсистемой GUI;

1.2. Описание работы 1 Создайте карту навигации для выбранной системы. На карте в зависимости от специфики системы выделите разделы, доступные различным пользователям в зависимости от роли, опишите условия перехода из различных разделов (при необходимости) 2 Используя графический редактор на выбор, создайте макеты графического интерфейса пользователя (от каждого члена бригады – не менее 3 макетов). Предлагаемые системы: - Microsoft Visio 2010 - Axure - Adobe Photoshop - Balsamiq – Cacoo

3 Для разработанных макетов подготовьте их текстовое описание в следующем виде:

Название поля

Тип

Условия видимости

Условия доступности

Описание

Формат, допустимые значения, макс. и мин. длина, поведение 1.3. Содержание отчета

1 Титульный лист

2 Цель работы

3 Навигационная схема (карта навигации)

4 Макеты графического интерфейса пользователя

5 Описание элементов управления по таблице 1

6 Выводы

**Контрольные вопросы**

1 Что такое графический интерфейс пользователя?

2 Какие бывают виды графического интерфейса?

3 Что такое карта навигации?

**Практическое занятие № 25 «Выполнение функционального тестирования»**

**Цель:** изучить подробно средства автоматизации тестов, на примере IBM Rational Functional Tester, IBM Rational Performance Tester, TestComplete 10.

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

**Краткая теория и ход выполнения работы**

IBM Rational Functional Tester IBM Rational Functional Tester - это инструмент автоматического функционального тестирования и регрессионного тестирования. Это программное обеспечение предоставляет функции автоматического тестирования для функционального, регрессионного тестирования, тестирования графических пользовательских интерфейсов и тестирования, ориентированного на данные. Rational Function Tester поддерживает ряд приложений, таких как веб-приложения, приложения для .Net, Java, Siebel, SAP, приложения на основе эмулятора терминала, PowerBuilder, Ajax, Adobe Flex, Dojo Toolkit, GEF, документы Adobe PDF, zSeries, iSeries и pSeries. IBM Rational Performance Tester Performance Tester - это инструмент тестирования производительности, с помощью которого можно выявлять проблемы системной производительности и их причины. Rational Performance Tester предоставляет тестировщикам средства автоматизированного тестирования, позволяющие выполнять функциональное тестирование, регрессивное тестирование, тестирование пользовательского интерфейса и тестирование, управляемое данными. IBM Rational Quality Manager Rational Quality Manager — это решение для совместной работы, предназначенное для обеспечения высокого качества программного обеспечения и систем с учетом особенностей бизнеса и поддерживающее практически все платформы и типы тестирования. Это программный продукт обеспечивает удобное совместное использование данных в рабочих группах, поддерживает автоматизацию для сокращения сроков выполнения проектов и позволяет создавать отчеты о ходе работы для принятия более обоснованных решений. IBM Rational Test RealTime IBM Rational Test RealTime - межплатформное решение, обеспечивающее тестирование компонентов и анализ их работы. Создано специально для разработчиков сложных систем встроенных приложений, приложений реального времени и других типов межплатформных программных продуктов. Это ПО позволяет отлаживать и исправлять ошибки до того, как они попадут в программный код готового продукта. TestComplete 10 TestComplete — это инструмент для автоматизации тестирования, позволяющий Вам создавать и выполнять тесты для Windows, .NET, WPF, Visual C++, Visual Basic, Delphi, C++ Builder, Java и веб-приложений, как простых, так и обладающих богатой функциональностью. TestComplete ориентирован как на функциональное так и модульное тестирование. Используя TestComplete, Вы с легкостью сможете создавать и автоматизировать тесты для ваших приложений. Новые возможности TestComplete 10: • поддержка Microsoft Visual Studio 2013 и Embarcadero RAD Studio XE7 • поддержка Windows 8.1 • Поддержка Ready! API и SoapUI. TestExecute TestExecute - это утилита, предназначенная для выполнения TestComplete проектов на компьютерах, где не установлен TestComplete. TestExecute дает тестировщикам возможность демонстрировать работу тестов в реальных условиях – например, на машине клиента с учетом особенностей его программного окружения – и освобождает от необходимости устанавливать TestComplete на целевые компьютеры. TestExecute удобно использовать для распределенного тестирования. Borland SilkTest SilkTest инструмент функционального тестирования приложений. Дает тестировщикам программного обеспечения возможность легко успевать за темпами разработки. Silk Test разработан в расчете на интеграцию со средой разработки, он обеспечивает постоянные и воспроизводимые результаты тестов в условиях популярных сегодня укороченных жизненных циклов выпуска продукта. Silk Test поддерживает гибкие циклы разработки, помогая тщательно проверять надежность приложений при помощи сложных тестов. IBM Rational Application Performance Analyzer IBM Rational Application Performance Analyzer позволяет пользователям выявлять программный код приложения, который является причиной низкой производительности. IBM Rational Performance Test Pack Virtual Testers IBM Rational Performance Tester - это инструмент тестирования производительности, с помощью которого можно выявлять проблемы системной производительности и их причины. IBM Rational Performance Tester предоставляет тестировщикам средства автоматизированного тестирования, позволяющие выполнять функциональное тестирование, регрессивное тестирование, тестирование пользовательского интерфейса и тестирование, управляемое данными. IBM Rational Performance Test Server IBM Rational Performance Test Server — это средство тестирования рабочей нагрузки, которое позволяет проверить производительность и масштабируемость приложения. Решение IBM Rational Performance Test Server максимизирует инфраструктуру тестирования для быстрого развертывания сценариев загрузки и проведения масштабного тестирования системы. IBM Rational Performance Tester Extension for Siebel Test Automation Rational Performance Tester Extension for Siebel Test Automation это быстрый поиск и устранение неисправностей за счет проверки типов данных Siebel. IBM Rational Performance Tester Extension for Siebel Test Automation расширяет возможности IBM Rational Performance Tester за счет предоставления средств тестирования производительности и работы при различных нагрузках для приложений Siebel 7.7. Сочетая удобные в использовании функции с глубокой детализацией, Rational Performance Tester Extension for Siebel Test Automation упрощает разработку тестов, моделирование нагрузки и процессы сбора данных, что позволяет полностью проверить готовность приложений к продуктивному использованию. IBM Rational Performance Tester for z/OS IBM Rational Performance Tester for z/OS - это единственное средство для тестирования производительности под нагрузкой, позволяющее использовать оборудование серии zSeries для моделирования нагрузки. Rational Performance Tester for z/OS сочетает в себе множество удобных функций с широкими возможностями для тестирования, которые упрощают создание тестов, моделирование нагрузки и анализ. Это помогает разработчикам создавать приложения, способные справляться с требуемой нагрузкой. IBM RATIONAL Service Tester For Soa Quality IBM Rational Service Tester for SOA Quality - это инструмент обеспечения качества SOAприложений, построенных на основе Web-служб. Предоставляет тестировщикам возможности автоматического тестирования без сценариев, обеспечивающие выполнение функционального тестирования, регрессивного тестирования и тестирования производительности Web-служб без пользовательского интерфейса. IBM Rational Test Virtualization Server IBM Rational Test Virtualization Server решение, помогающие оптимизировать качество проекта с помощью центра объединенного управления тестированием, предоставляющего интегрированную поддержку жизненного цикла практически для любой платформы и типа тестирования. IBM Rational Test Virtualization Server позволяет создавать, изменять и разворачивать приложения для упрощенного, эффективного тестирования. Это решение ускоряет доставку сложных сред тестирования и позволяет выполнять тестирование интеграции раньше и более часто в цикле разработки. IBM Rational Test Workbench IBM Rational Test Workbench средство для комплексного тестирования функций, регрессии, загрузки и интеграции. IBM Rational Test Workbench создавает интеллектуальные и взаимосвязанные приложения предприятий, которые могут быть развернуты в традиционных и облачных инфраструктурах. IBM Rational Test Workbench сокращает время циклов тестирования и раньше переносить тестирование интеграции в жизненном цикле разработки. SmartBear Collaborator (ранее: CodeCollaborator Code) SmartBear Collaborator – это лучший коммерческий инструмент для критического анализа кода разрабатываемых программ. Также Collaborator предоставляет возможность анализировать изображения, проектную документацию, документы Microsoft Office гораздо быстрее, чем обычно. Критический анализ кода - это лучший способ повысить качество разрабатываемого программного обеспечения. Благодаря Collaborator проверка кода становится еще эффективнее. Collaborator упрощает и ускоряет процесс анализа кода, предоставляя подробные отчеты, что поможет улучшить Ваши программы. AQtime Pro AQtime - это инструмент для повышения производительности и улучшения качества приложений. AQtime может анализировать 32 и 64-ех разрядные Windows, .NET, Silverlight и Java приложения, созданные с помощью C#, VB.NET, Visual C++, Visual Basic, Delphi, C++Builder, Intel C++, Compaq Visual Fortran и GNU C++ компиляторов. AQtime также поддерживает работу с JScript и VBScript кодом. AQtime интегрируется в Visual Studio, а также в Embarcadero RAD Studio, что позволяет находить узкие места и оптимизировать ваши программы, не покидая среды разработки. Задание для самостоятельной работы

1. Изучить теоретически существующие средства автоматизации тестов.

2. Законспектировать кратко теорию по лабораторной работе.

3. Ознакомится самостоятельно на практике со следующими средствами автоматизации тестов: IBM Rational Functional Tester, IBM Rational Quality Manager, TestComplete 10. Для этого воспользоваться ярлыками программ на рабочем столе ПК. 4. Оформить отчет и вывод о проделанной работе.

Отчет должен содержать:

1. № и тему лабораторной работы.

2. Выполненное задание по лабораторной работе.

**Практическое занятие № 26 «Тестирование интеграции»**

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

**Тестирование интеграции**

Тестирование интеграции поддерживает сборку цельной программной системы. **Цель сборки и тестирования интеграции:** взять модули, протестированные как элементы, и построить программную структуру, требуемую проектом. Тесты проводятся для обнаружения ошибок интерфейса. Перечислим некоторые категории ошибок интерфейса:

* Потеря данных при прохождении через интерфейс;
* Отсутствие в модуле необходимой ссылки;
* Неблагоприятное влияние одного модуля на другой;
* Подфункции при объединении не образуют требуемую главную функцию;
* Отдельные неточности при интеграции выходят за допустимый уровень;
* Проблемы при работе с глобальными структурами данных.

Существует два варианта тестирования, поддерживающих процесс интеграции: нисходящее тестирование и восходящее тестирование.

**Нисходящее тестирование интеграции.**В данном подходе модули объединяются  движением сверху вниз по управляющей иерархии, начиная от главного управляющего модуля. Подчиненные модули добавляются в структуру или в результате поиска в глубину, или в результате поиска в ширину. Рассмотрим пример (рис.8.4).

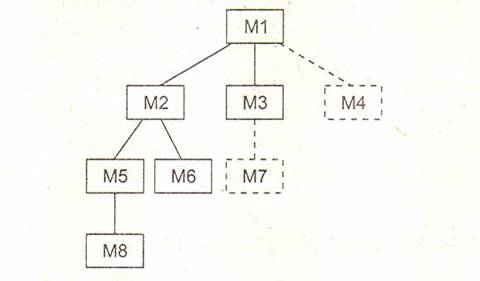


Рисунок 8.4 – Нисходящая интеграция системы

Опишем шаги процесса нисходящей интеграции.

1. Главный управляющий модуль(находится на вершине иерархии) используется как  тестовый драйвер. Все непосредственно подчиненные ему модули временно замещаются заглушками.
2. Одна из заглушек заменяется реальным модулем. Модуль выбирается поиском в ширину или в глубину.
3. После подключения каждого модуля (и установки на нем заглушек) проводится набор тестов, проверяющих полученную структуру.
4. если в модуле – драйвере уже нет заглушек, производится смена модуля-драйвера (поиском в ширину и ли глубину).
5. Выполняется возврат на шаг 2 (до тех пор, пока не будет построена целая структура).

Категории заглушек представлены на рис.8.5.

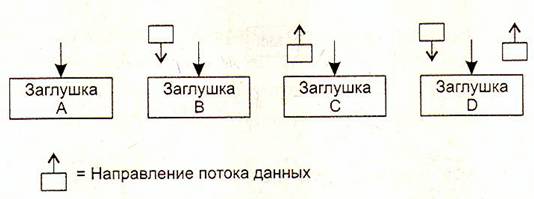


Рисунок 8.5 – Категории заглушек

**Восходящее тестирование интеграции.** При восходящем тестировании интеграции сборка и тестирование системы начинается с модулей – атомов, располагаемых на нижних уровнях иерархии. Модули подключаются движением снизу вверх. Подчиненные модули всегда доступны, нет необходимости в заглушках.

            Рассмотрим по шагам методику восходящей интеграции:

1. Модули нижнего уровня объединяются в кластеры(группы, блоки), выполняющие определенную программную функцию.
2. Для координации вводов-выводов тестового варианта пишется драйвер, управляющий тестированием кластеров.
3. Тестируется кластер.
4. Драйверы удаляются, а кластеры объединяются в структуру движением вверх.

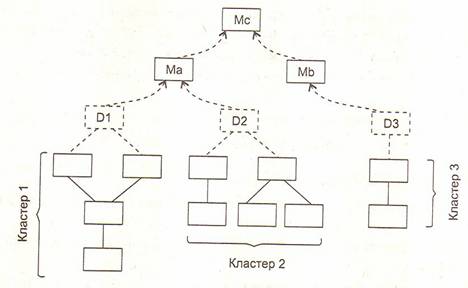


Рисунок 8.6 – Восходящие интеграция системы

Пример восходящей интеграции системы приведен на рис.8.6.

Различные типы драйверов приведены на рис.8.7.



Рисунок 8.7 – Различные типы драйверов

**Тестирование правильности**

После окончания тестирования интеграции программная система собрана в единый корпус, интерфейсные ошибки обнаружены и откорректированы. Теперь начинается последний шаг программного тестирования – тестирование правильности. Цель – подтвердить, что функции , описанные в спецификации требований к ПС, соответствуют ожиданиям заказчика.

            Подтверждение правильности ПС выполняется с помощью тестов «черного ящика», которые демонстрируют  соответствие требованиям. При обнаружении отклонений от спецификации требований создается список недостатков. Как правило, отклонения и ошибки, выявленные при подтверждении правильности, требуют изменения сроков разработки продукта.

            Важным элементом подтверждения правильности является проверка конфигурации ПС. Конфигурацией ПС называют совокупность всех элементов информации, вырабатываемых в процессе конструирования ПС. Минимальная конфигурация ПС включает следующие базовые элементы:

1. системную спецификацию;
2. план программного проекта;
3. спецификацию требований к ПС, работающий или бумажный макет;
4. предварительное руководство пользователя;
5. спецификация проектирования;
6. листинги исходных текстов программ;
7. план и методику тестирования, тестовые варианты и полученные результаты;
8. руководство по работе и инсталляции;
9. exe-код выполняемой программы;
10. описание базы данных;
11. руководство пользователя по настройке;
12. документы сопровождения;
13. стандарты и методики конструирования ПС.

**Системное тестирование**

Системное тестирование подразумевает выход за рамки области действия программного проекта и проводится не только программным разработчиком. Классическая проблема системного тестирования – указание причины. Она возникает, когда разработчик одного системного элемента обвиняет разработчика другого элемента в причине возникновения дефекта. Для защиты от подобного обвинения разработчик программного элемента должен:

1. предусмотреть средства обработки ошибок, которые тестируются все вводы информации от других элементов системы;
2. провести тесты, моделирующие неудачные данные или другие потенциальные ошибки интерфейса ПС;
3. записать результаты тестов, чтобы использовать их как доказательство невиновности в случае «указания причины»;
4. принять участие в планировании и проектировании системных тестов, чтобы гарантировать адекватное тестирование ПС.

**Тестирование восстановления**

Многие компьютерные системы должны восстанавливаться после отказов и возобновлять обработку в пределах заданного времени. В некоторых случаях система должна быть отказоустойчивой, то есть отказы обработки не должны быть причиной прекращения работы системы. В других случаях системный отказ должен быть устранен в пределах заданного кванта времени, иначе заказчику наносится экономический ущерб.

      Тестирование восстановления используют разные пути для того, чтобы заставить ПС отказать, и проверяют полноту выполненного восстановления. При автоматическом восстановлении оценивается правильность повторной инициализации, механизм копирования контрольных точек, восстановление данных, перезапуск. При ручном оценивается , находится ли среднее время в допустимых пределах.

**Тестирование безопасности**

Компьютерные системы очень часто являются мишенью незаконного проникновения. Под проникновением понимается широкий диапазон действий: попытки хакеров проникнуть в систему , месть рассерженных служащих, взлом мошенниками для незаконной наживы.

      Тестирование безопасности проверяет фактическую реакцию защитных механизмов, встроенных в систему, на проникновение.

В ходе тестирования безопасности испытатель играет роль взломщика. Ему разрешено все:

* попытки узнать пароль с помощью внешних средств;
* атака системы с помощью специальных утилит, анализирующих защиты;
* подавление, ошеломление системы;
* целенаправленное введение ошибок в надежде проникнуть в систему в ходе восстановления;
* просмотр несекретных данных в надежде найти ключ для входа в систему.

Задача проектировщика системы – сделать цену проникновения более высокой, чем цена получаемой в результате информации.

**Стрессовое тестирование**

На предыдущих шагах тестирования способы «белого » и «черного ящиков» обеспечивали полную оценку нормальных программных функций и качества функционирования. Стрессовые тесты проектируются для навязывания программам ненормальных ситуаций. В сущности, проектировщик стрессового теста спрашивает, как сильно можно расшатать систему, прежде чем она откажет?

                  Стрессовое тестирование производится при ненормальных запросах на ресурсы системы(по количеству, частоте, размеру-объему).

По существу испытатель разрушает систему. Разновидность стрессового тестирования называется тестированием  **чувствительности.**Тестирование чувствительности обнаруживает комбинации данных, которые могут вызвать нестабильность или неправильность обработки.

**Тестирование производительности**

                  В системах реального времени и встроенных системах недопустимо ПО, которое реализует требуемые функции, но не соответствует требованиям производительности.

                  Тестирование производительности проверяет скорость работы ПО в компьютерной системе. Производительность тестируется на всех шагах процесса тестирования. Даже на уровне элемента при проведении тестов «белого ящика» может оцениваться производительность индивидуального модуля. Иногда тестирование производительности сочетают со стрессовым тестированием. При этом нередко требуется специальный     аппаратный и программный инструментарий. Например, часто требуется точное измерение используемого ресурса(процессорного цикла). Внешний инструментарий регулярно отслеживает интервалы выполнения, регистрирует события(например, прерывания) и машинные состояния, которые приводят к деградации и возможным отказам системы.

**Искусство отладки**

**Отладка –** это локализация и устранение ошибок. Отладка является следствием успешного тестирования. Это значит, что если тестовый вариант обнаруживает ошибку, то процесс отладки уничтожает ее.

            Процессу отладки предшествует выполнение тестового варианта. Его результаты  оцениваются, регистрируется несоответствие между ожидаемым  и реальным результатами. Несоответствие является симптомом скрытой причины. Процесс отладки пытаются сопоставить  с причиной, вследствие чего ошибка исправляется. Возможны два исхода процесса отладки:

1. причина найдена, исправлена, уничтожена;
2. причина не найдена.

Во втором случае отладчик может предполагать причину. Для проверки этой причины он просит разработать дополнительный тестовый вариант, который поможет проверить предположение. Таким образом, запускается итерационный процесс корректировки ошибки.

            Возможны разные способы проявления ошибок:

1. программа завершается нормально, но выдает неверные результаты;
2. программа зависает;
3. программа завершается по прерыванию;
4. программа завершается, выдает ожидаемые результаты, но хранимые данные испорчены.

В ходе отладки мы встречаем  ошибки в широком диапазоне: от мелких неприятностей до катастроф.  Английский  термин debugging(отладка) дословно переводиться как «ловля блох», который отражает специфику процесса – погоню за объектами отладки. Различают две группы методов отладки:

* аналитические;
* экспериментальные.

Аналитические методы базируются на анализе выходных данных для тестовых прогонов. Экспериментальные  методы базируются на использовании вспомогательных средств отладки(отладочные печати, трассировки), позволяющих уточнить характер проведения программы при тех или иных исходных данных.

            Общая стратегия  отладки – обратное прохождение от замеченного симптома ошибки к исходной аномалии(месту в программе , где ошибка совершена).

Цель отладки – найти оператор программы, при исполнении которого правильные аргументы приводят к неправильным результатам. Если место проявления симптомы ошибки не является искомой аномалией, то один  из аргументов оператора должен быть неверным. Поэтому надо перейти к исследованию предыдущего оператора, выработавшего этот неверный аргумент. В итоге пошаговое обратное прослеживание приводит к искомому ошибочному месту.

            В разных методах прослеживание организуется по-разному. В аналитических методах – на основе логических заключений о поведении программы. Цель – шаг за шагом уменьшить область программы, подозреваемую в наличии ошибки. Здесь определяется корреляция между значениями выходных данных и особенностями поведения.

            Основное преимущество аналитических методов отладки состоит в том, что исходная программа остается без изменений.

            В экспериментальных методах отладки для прослеживания выполняется:

* выдача значений переменных в указанных точках;
* трассировка переменных;
* трассировка потоков управления.

**Контрольные вопросы:**

1. В чем состоит суть методики тестирования ПС?
2. Когда выполняется тестирование элементов?
3. Какой этап конструирования проверяет системное тестирование?
4. Что такое драйвер тестирования?
5. Что такое заглушка?
6. Что такое отладка?
7. В чем суть аналитических методов отладки?
8. Какие элементы включает минимальная конфигурация программной системы?
9. В чем суть регрессионного тестирования?
10. Что такое тестирование правильности?

**Практическое занятие № 27 «Документирование результатов тестирования»**

**Цель работы**: составить итоговый отчет о результатах тестирования web-приложения.

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

**Теоретические сведения**

Если во время тестирования не произошло никаких форс-мажорных обстоятельств, то достаточно обычного вежливого приветствия и далее уже переход к следующим пунктам. Общая информация (Common Information) В данной части отчета описывается, какие виды тестов проводились. Зачастую указываются модули, которые тестировались или функционал. Стоит удостовериться, не забыта ли какая-то часть функционала, особенно это актуально, когда нужно собрать итоговый отчет, соединив в себе данные о разных видах тестов и функционале. Тестовое окружение (Test Platform) Как правило, в этой части указываются: −Название проекта. −Номер сборки. −Ссылка на проект (сборку). Необходимо убедиться, что зайдя по этой ссылке вы действительно попадаете на проект или можете установить приложение. При указании данных в этой части отчета нужно быть очень внимательным, т.к. неправильная ссылка на сборку или неверный номер сборки не дают достоверной информации всем заинтересованным людям, а также затрудняют работу человеку, собирающему финальный отчет.

Также как и у предыдущего вида тестов, качество этих может быть: Высокое, Среднее, Низкое. Для указанных видов тестов в данной части отчета должна быть описана информация следующего характера:

−Дана сравнительная характеристика каждой из частей функционала в сравнении с предыдущими версиями сборки.

−Подробная (детальная) информация о качестве каждой из частей проверенной функциональности.

−Даны ясные пояснения о выставлении соответствующего качества каждой функции в отдельности.

−Даны рекомендации по улучшению качества (какие проблемы следует исправить). Окончание содержимого.

В завершении содержимое отчета должно включать в себя информацию следующего характера:

−Ссылка на тест-план.

−Ссылка на документ feature matrix (если таковой имеется).

−Ссылка на документ со статистикой (если таковой имеется).

−Общее количество всех новых дефектов.

−Подпись высылающего отчет. Данные ссылки должны быть корректными, необходимо проверить достоверную ли информацию получает пользователь, открывший ссылку. Следует обращать особое внимание на подпись, удостоверьтесь, что указана именно ваша подпись либо какая-то универсальная для определенного проекта подпись.

Порядок выполнения работы

1. Получить задание у преподавателя.

2. Составить итоговый отчет по результатам тестирования webприложения.

3. Оформить отчет и защитить лабораторную работу.

Содержание отчета

1. Цель работы.

2. Краткие теоретические сведения.

3. Итоговый отчет о результатах тестирования web-приложения.

4. Выводы по работе.

Контрольные вопросы

1. Какая структура итогового отчета о результатах тестирования?

2. Что содержится в разделе Приветствие?

3. Что содержится в разделе Общая информация?

**Практическое занятие № 28 «Построение простейших математических моделей. Построение простейших статистических моделей»**

**Цель работы:** закрепить практические навыки по построению простейших математических и простейших статистических моделей.

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

## Краткая теория

Построение математической модели процесса, явления или объ­екта начинается с построения упрощенного варианта модели, в ко­тором учитываются только основные черты. В результате прослеживаются основные связи между входными параметрами, ограничениями и показателем эффективности. Общего подхода к построе­нию модели нет. В каждом конкретном случае при построении математической модели учитывается большое количество факторов: цель построения модели, круг решаемых задач, точность описания модели и точность выполнения вычислений. Математическая модель должна отражать все существенные факторы, определяющие ее по­ведение, и при этом быть простой и удобной для восприятия резуль­татов. Каждая математическая модель процесса, явления или объекта в своей основе имеет математический количественный метод.

Применение математических количественных методов для обоснования выбора того или иного управляющего решения во всех областях человеческой деятельности называется *исследованием операций.*Целью исследования операций является нахождение с использованием специального математического аппарата решения, удовлетворяющего заданным условиям. На самом деле при реше­нии практически любой задачи имеется неограниченное количество решений. Множество решений, удовлетворяющих заданным усло­виям (ограничениям), называется допустимым множеством решением. Выбор из множества допустимых решений одного решения, наилучшего в каком-либо смысле, называемого *оптимальным*решением, и есть задача исследования операций.

*Модель — это материальный или идеальный объект, заменяю­щий оригинал, наделенный основными характеристиками (черта­ми) оригинала и предназначенный для проведения некоторых дей­ствий над ним с целью получения новых сведений об оригинале.*



Рис. 1. Классификация моделей

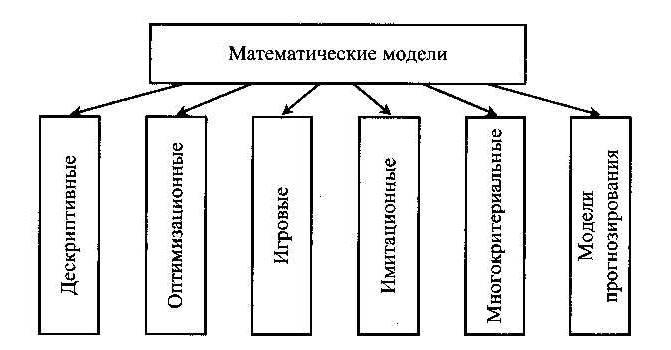


Рис. 2. Классификация математических моделей

При построении математической модели необходимо обеспе­чить *достаточную*точность вычислений (точность решения) и *не­обходимую*подробность модели. Любая математическая модель включает в себя описание основных, т. е. *необходимых*для исследо­вания свойств и законов функционирования исследуемого объекта, процесса или явления. В своей основе каждая мате­матическая модель имеет целевую функцию, которая описывает функционирование реального объекта, процесса или явления. В зависимости от исследуемого (моделируемого) объекта, явления или процесса *целевая функция*может быть представлена одной функ­циональной зависимостью, системой уравнений (линейных, нели­нейных, дифференциальных и т. д.), набором статистических дан­ных и т. д. При работе с целевой функцией исследователь воздейст­вует на нее через ***набор входных параметров***(рис. 3).

|  |  |  |
| --- | --- | --- |
| Входной параметр 1 |  | Выходной параметр 1 |
| Входной параметр 2 |  | Выходной параметр *2* |
| Входной параметр 3 | Модель системы | Выходной параметр 3 |
| Входной параметр *п-*1 | (объекта или процесса) | Выходной параметр *т -*1 |
| Входной параметр*n* |  | Выходной параметр *т* |
|  |  |  |

Рис. 3. Обобщенная схема математической модели

По способу реализации математические модели можно разде­лить следующим образом.

1. Линейное программирование.

Математическая модель целиком (целевая функция и ограниче­ния) описывается уравнениями первого порядка. Линейное програм­мирование включает в себя несколько методов решения (задач):

* симплексный;
* графический;
* транспортная задача;
* целочисленное программирование.

2. Нелинейное программирование.

Целевая функция и ограничения, составляющие математическую модель, содержат хотя бы одно нелинейное уравнение (уравнение второго порядка и выше). Нелинейное программирование содержит несколько методов решения (задач):

* графический;
* регулярного симплекса;
* деформируемого многогранника (Нелдера - Мида);
* градиентный.

3. Динамическое программирование.

Ориентировано на решение задач прокладки магистралей крат­чайшим путем и перераспределения различных видов ресурсов.

4. Сетевое планирование.

Решает проблему построения графика выполнения работ, рас­пределения производственных, финансовых и людских ресурсов.

5. Принятие решений и элементы планирования.

В этом случае и качестве целевой функции выступает набор ста­тистических данных или некоторые данные прогноза. Решением задачи являются рекомендации о способах поведения (стратегии). Решение носит рекомендательный характер (приблизительное решение). Выбор стратегии целиком остается за человеком — ответ­ственным лицом, принимающим решение. Для принятия решения разработаны следующие теории:

* теория игр;
* системы массового обслуживания.

**Теоретическая часть**

**Задача 1.** Составить математическую модель следующей задачи. Предположим, что для производства продукции вида А и В можно использовать материал трех сортов. При этом на изготовление единицы изделия вида А расходуется ***а1*** кг первого сорта, ***а2*** кг второго сорта и ***а3*** кг третьего сорта. На изготовление продукции вида В расходуется ***b1*** кг первого сорта, ***b2*** кг второго сорта, ***b3*** кг третьего сорта. На складе фабрики имеется всего материала первого сорта ***с1*** кг, второго сорта ***с2*** кг, третьего сорта ***с3*** кг. От реализации единицы готовой продукции вида А фабрика имеет прибыль вида ***α*** руб., а от реализации единицы готовой продукции вида В фабрика имеет прибыль вида ***β*** руб. Определить максимальную прибыль от реализации всей продукции видов А и В.

***а1= 19, а2= 16, а3= 19, b1= 26, b2= 17, b3= 8, c1= 868, c2= 638, c3= 853,***

***α=5, β=4.***

**Задача 2.** Имеются три пункта поставки однородного груза **А1, А2, А3** и пять пунктов **В1, В2, В3, В4, В5**потребления этого груза. На пунктах **А1, А2 и А3** находится груз соответственно в количестве ***а1, а2 и а3*** тонн. В пункты **В1, В2, В3, В4, В5** требуется доставить соответственно ***b1, b2, b3, b4, b5*** тонн груза. Расстояние между пунктами поставки и пунктами потребления приведено в таблице:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Пункты поставки** | **Пункты потребления** | | | | |
| **В1** | **В2** | **В3** | **В4** | **В5** |
| **А1** | D11 | D12 | D13 | D14 | D15 |
| **А2** | D21 | D22 | D23 | D24 | D25 |
| **А3** | D31 | D32 | D33 | D34 | D35 |

Найти такой план закрепления потребителей за поставщиками однородного груза, чтобы общие затраты по перевозкам были минимальными.

|  |  |
| --- | --- |
| ***а1=300, а2=250, а3=200,***  ***b1=210, b2=150, b3=120, b4=135,b5=135.*** |  |

**Практическое занятие № 29 «Решение простейших однокритериальных задач»**

**Цели занятия**:

1. Отработать и закрепить умения графически решать системы неравенств с двумя переменными.

2. Отработать и закрепить умения записывать взаимосвязь показателей задачи в виде математической модели.

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

**Методические указания к выполнению заданий практического занятия**

*Решение системы неравенств с двумя переменными графическим методом включает следующие этапы.*

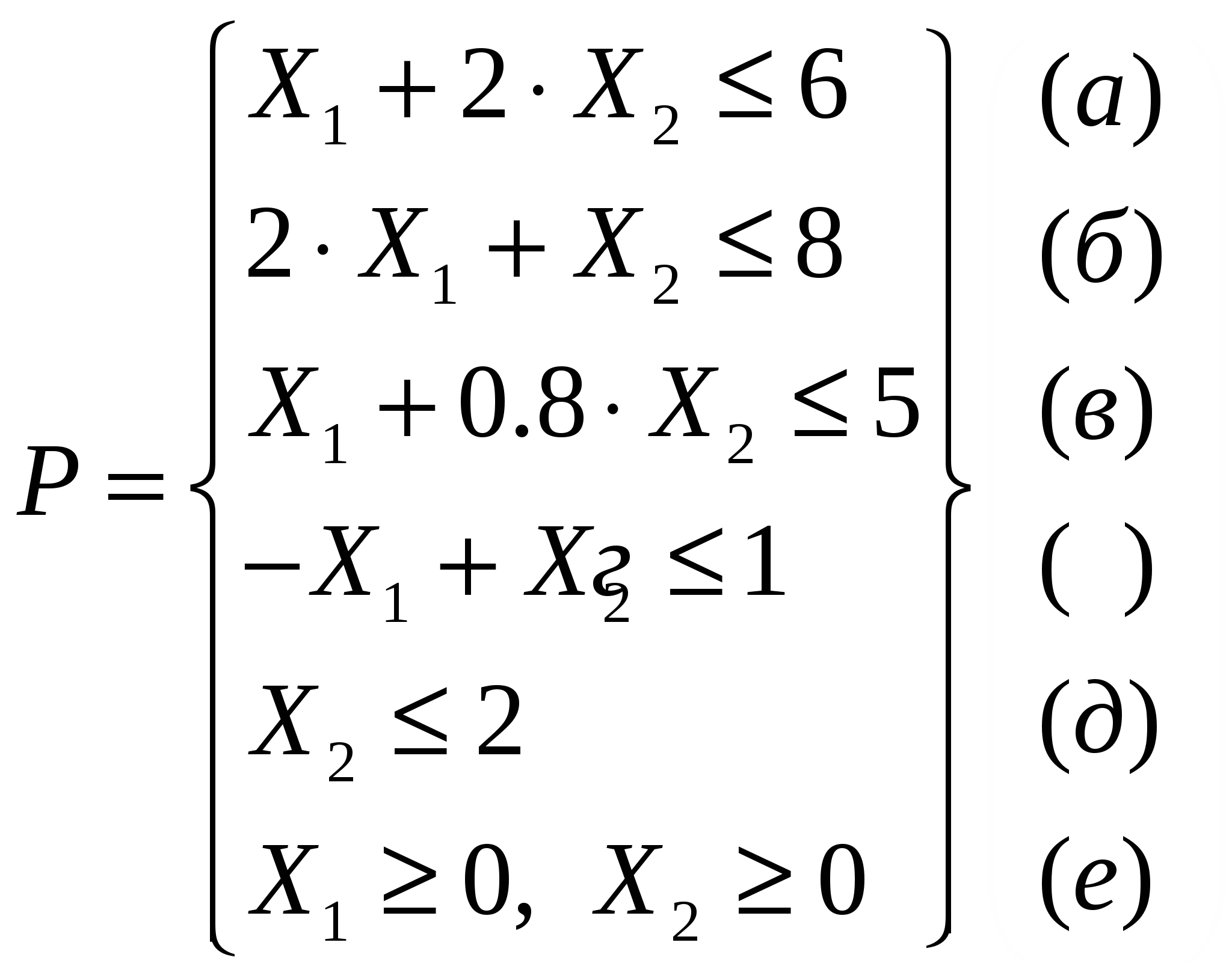
1. На плоскости Х1ОХ2 строят прямые, уравнения которых получаются в результате замены в ограничениях знаков неравенств на знаки точных равенств.

2. Находят полуплоскости, определяемые каждым из неравенств.

3. Строят многоугольник решений.

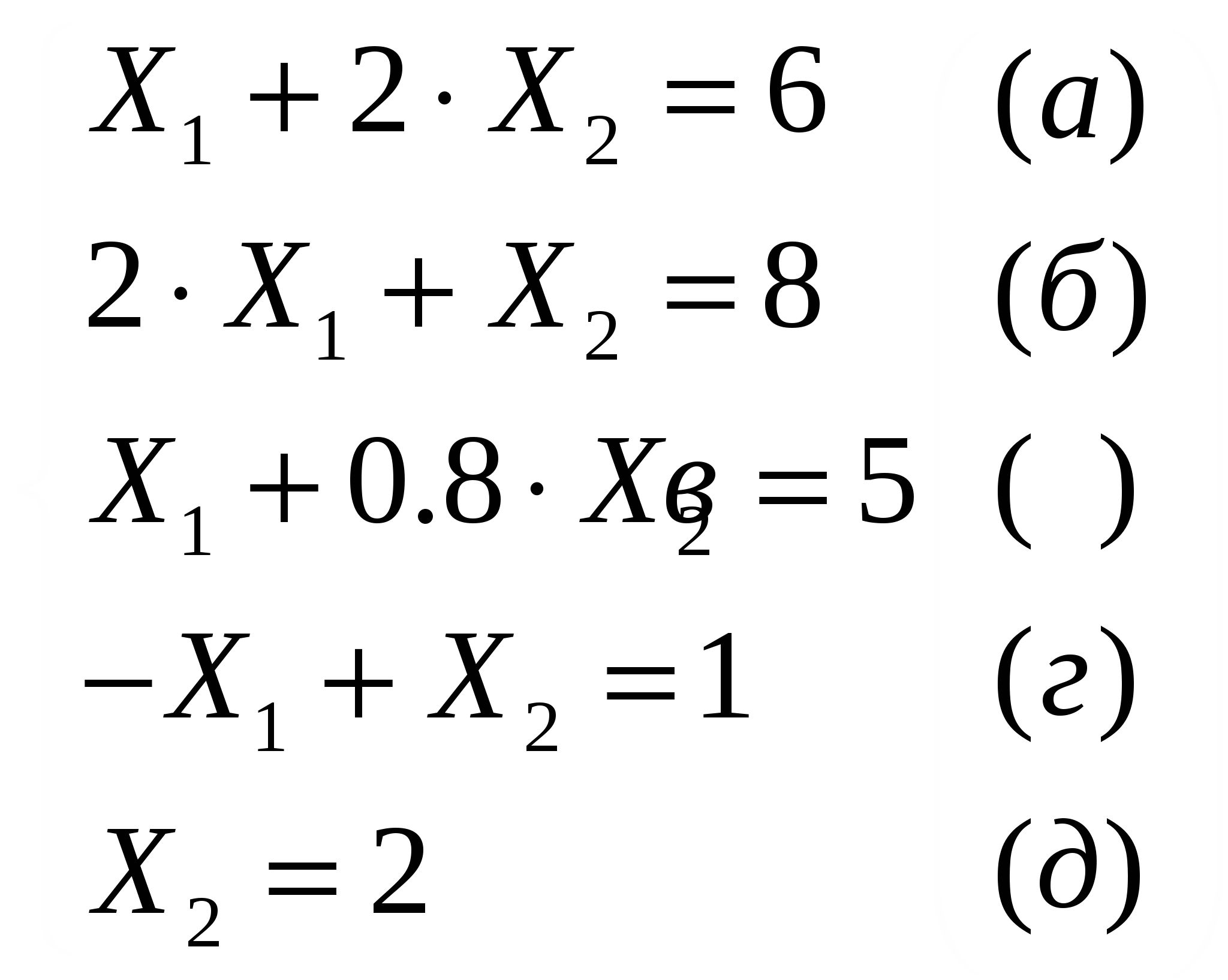
**Пример:**

Решить систему неравенств графическим способом.



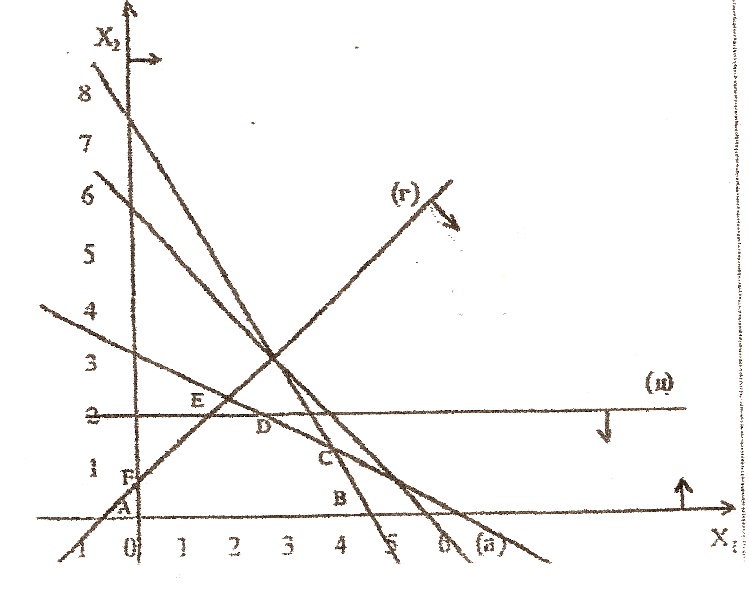
**Решение:**

Шаг 1. Строим область допустимых решений - область Р, т.е. геометрическое место точек, в котором одновременно удовлетворяются все ограничения ЗЛП. Каждое из неравенств (а)-(д) системы ограничений задачи геометрически определяет полуплоскость соответственно с граничными прямыми:



Условия неотрицательности переменных (е) ограничивают область допустимых решений первым квадратом. Области, в которых выполняются соответствующие ограничения в виде неравенств, указываются стрелками, направленными в сторону допустимых значений переменных

Решение системы – многоугольник ABCDEF.



**Практическая часть**

1. Решить графически систему неравенств:

а) x1+ x2≤ 5

3x1- x2≤ 3

x1≥0, x2≥0

б) x1+ x2≤ 4

6x1+ 2x2≥ 6

x1+ 5x2≥ 5

x1≥0, x2≥0

1. Составить математическую модель задачи и найти решение системы ограничений:

Чулочно-носочная фирма производит и продает два вида товаров: мужские носки и женские чулки. Фирма получает прибыль в размере 10 руб. от производства и продажи одной пары чулок и в размере 4 руб. от производства и продажи одной пары носков. Производство каждого изделия осуществляется на трех участках. Затраты труда (в часах) на производство одной пары указаны в следующей таблице для каждого участка:

Участок производства

Чулки

Носки

1

0,02

0,01

2

0,03

0,01

3

0,03

0,02

Руководство рассчитало, что в следующем месяце фирма ежедневно будет располагать следующими ресурсами рабочего времени на каждом из участков: 60 ч на участке 1; 70 ч на участке 2 и 100 ч на участке 3. Сколько пар носков и чулок следует производить ежедневно, если фирма хочет максимизировать прибыль?

**Практическое занятие № 30 «Задача Коши для уравнения теплопроводности»**

**Цели занятия**: отработать и закрепить умения решать Задачи Коши для уравнения теплопроводности.

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

**Теоретическая часть**

Начнем изучение задачи Коши для уравнения теплопроводности во всем пространстве по 𝑥 ∈ R 1 . В этом случае нет граничных условий, и задается только начальное распределение температуры. Обозначим через 𝐷 полосу {𝑥 ∈ R 1 , 0 < 𝑡 ≤ 𝑇} на плоскости (𝑥, 𝑡). Тогда 𝐷¯ = {︀ 𝑥 ∈ R 1 , 0 ≤ 𝑡 ≤ 𝑇 }︀ . Уравнение теплопроводности имеет вид 𝐿𝑢 := 𝜕𝑢 𝜕𝑡 − 𝑎 2 𝜕 2𝑢 𝜕𝑥2 = 0, 𝑡 ≥ 0, (1) а начальное условие 𝑢(𝑥, 0) = 𝜙(𝑥), 𝑥 ∈ R 1 . (2) Решением задачи (1), (2) будем называть ограниченную функцию 𝑢(𝑥, 𝑡) ∈ 𝐶 2 (𝐷)∩𝐶(𝐷¯), которая удовлетворяет уравнению (1) и начальному условию (2). Требование ограниченности решения означает, что для некоторого 𝑀 > 0 выполнено неравенство |𝑢(𝑥, 𝑡)| ≤ 𝑀, ∀(𝑥, 𝑡) ∈ 𝐷. Докажем, что решение задачи Коши в такой постановке является единственным и это решение непрерывно зависит от начальной функции 𝜙(𝑥). Пусть |𝜙(𝑥)| ≤ 𝜀, ∀𝑥 ∈ R 1

Зафиксируем произвольную точку (𝑥0, 𝑡0) ∈ 𝐷 и замкнутый прямоугольник 𝑄 = {(𝑥, 𝑡) | |𝑥| ≤ 𝑅, 0 ≤ 𝑡 ≤ 𝑇} , такой, что |𝑥0| < 𝑅, 0 < 𝑡0 ≤ 𝑇. Рассмотрим вспомогательную функцию 𝑤𝛿(𝑥, 𝑡) := 𝜀 + 𝛿(𝑥 2 + 4𝑎 2 𝑡) ± 𝑢(𝑥, 𝑡), где 𝛿 = 𝑀𝑅−2 . Тогда, очевидно, что 𝐿𝑤𝛿 = 𝛿(4𝑎 2 − 2𝑎 2 ) > 0. На нижнем основании прямоугольника 𝑄, где 𝑡 = 0, справедливо неравенство 𝑤𝛿 ≥ 0, а на боковой поверхности, где |𝑥| = 𝑅, 𝑤𝛿(±𝑅, 𝑡) ≥ 𝜀 + 𝛿𝑅2 − 𝑀 = 𝜀 > 0

в силу выбора числа 𝛿. Следовательно, из Леммы 1, доказанной на позапрошлой лекции, находим, что 𝑤𝛿(𝑥, 𝑡) ≥ 0, ∀(𝑥, 𝑡) ∈ 𝑄. Значит, |𝑢(𝑥0, 𝑡0)| ≤ 𝜀 + 𝛿(𝑥 2 + 4𝑎 2 𝑡0) = 𝜀 + 𝑀𝑅−2 (𝑥 2 + 4𝑎 2 𝑡0). В этом неравенстве, которое справедливо для всех 𝑅 > |𝑥0|, переходим к пределу при 𝑅 → ∞ и получаем оценку |𝑢(𝑥0, 𝑡0)| ≤ 𝜀, которая выполнена для любой точки (𝑥0, 𝑡0) ∈ 𝐷. Отсюда следует как единственность (ограниченного) решения 𝑢(𝑥, 𝑡), так и его непрерывная зависимость от 𝜙(𝑥) в равномерной sup-норме пространства 𝐶𝑏(R) непрерывных ограниченных функций.

**Практическая часть**

Найти решение уравнения

 (15.1)



для следующего распределения температуры стержня:

 (15.2)



**Практическая работа №31 «Сведение произвольной задачи линейного программирования к основной задаче линейного программирования»**

**Цель работы**: Научиться решать задачи линейного программирования графическим методом.

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

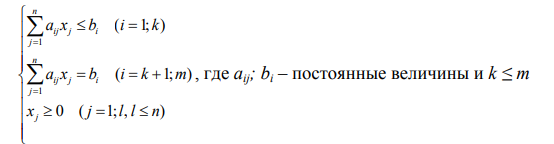
−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

**Содержание работы: Основные понятия.**

1 Математическое программирование — область математики, разрабатывающая теорию и численные методы решения многомерных экстремальных задач с ограничениями, т. е. задач на экстремум функции многих переменных с ограничениями на область изменения этих переменных.

2 В линейном программировании изучаются свойства решений линейных систем уравнений и неравенств с n переменными, называемых системами ограничений, следующего вида:



3 Система ограничений образует область допустимых решений (область экономических возможностей). План, удовлетворяющий системе ограничений задачи, называется допустимым.

4 Основной задачей линейного программирования (ОЗЛП) с n переменными называется задача о нахождении такого допустимого плана, который доставляет максимум (минимум) функции, называемой функцией прибыли (целевой функцией, показателем эффективности или критерием оптимальности).

5 Допустимый план, доставляющий максимум (минимум) целевой функции, называется оптимальным планом.

6 Целевая функция, экстремальное значение которой нужно найти в условиях экономических возможностей, система ограничений составляют математическую модель задачи — это отражение оригинала в виде функций, уравнений, неравенств, цифр и т. д.

7 Если хj рассматривать как координаты точки, то система ограничений является пересечением конечного числа полуплоскостей, образующее многоугольную область, которую называют областью решений системы неравенств.

**Практическая часть**

Для производства компьютерных столов I-го и II-го видов требуются три типа ресурсов: дерево, пластик и трудозатраты. Потребности в ресурсах для производства одного стола каждого вида, запасы ресурсов, а также прибыль от реализации одного стола каждого вида, заданы в следующей таблице: Тип ресурса Единица продукции I вида Единица продукции II вида Запас ресурса Дерево 1 3 24 Пластик 4 1 24 Трудозатраты 3 2 23 Прибыль 200 300 Требуется, решив задачу графическим методом, найти план выпуска продукции, позволяющий получить наибольшую прибыль.

**Практическая работа № 32«Составление систем уравнений Колмогорова. Нахождение финальных вероятностей. Нахождение характеристик простейших систем массового обслуживания»**

**Цель занятия:**

1. Отработать и закрепить умения составлять системы уравнений Колмогорова.
2. Отработать и закрепить умения находить финальные вероятности.
3. Отработать и закрепить умения определите основные показатели СМО.

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

**Методические указания к выполнению заданий практического занятия**

**Марковский случайный процесс**

Построение математических моделей в условиях неопределенности - очень сложнаяили невыполнимая задача. Лишь для некоторых упрощенных случаев можно построитьматематическую модель.

Следует различать два вида неопределенности:

* вероятностные характеристики либо известны, либо могут быть получены в результатеэксперимента. Такая неопределенность называется стохастической, и для большинстваобъектов, содержащих такую неопределенность, можно построить математическуюмодель, например выход из строя оборудования, приход нового клиента и т. д.
* вероятностные характеристики определить невозможно. В этом случае задачу можнопопытаться решить с помощью экспертных оценок, но результат будет весьмаприблизительным, например, каковы будут модели женской одежды через пять лет?

Строгую математическую модель с аналитическим вычислением всех интересующихвеличин можно построить только в том случае, если случайный процесс носитмарковский характер.

Случайный процесс будет марковским, если вероятностные характеристики процесса вмомент времени *t*зависят только от текущего (настоящего) состояния процесса в этотмомент времени *t*и не зависят от того, как (каким способом и когда) рассматриваемыйпроцесс перешел в текущее состояние.

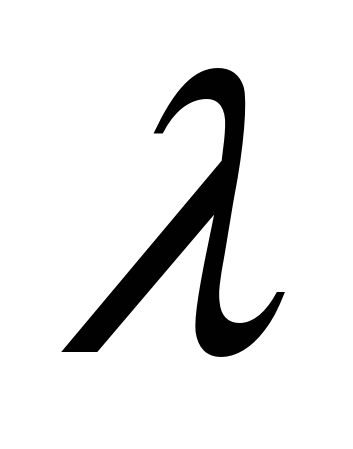
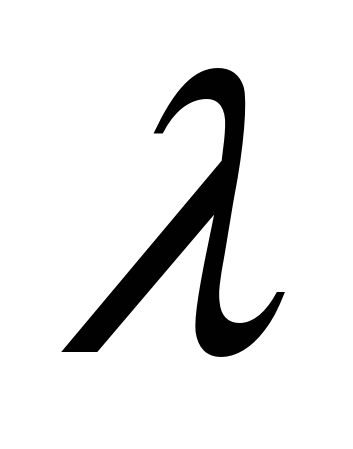
Из всего многообразия марковских процессов хорошо изучены и представляют большой практический интерес ***марковские случайные процессы с дискретнымисостояниями и непрерывным временем.***

Под дискретным состоянием будем понимать, что процесс переходит из одногосостояния в другое скачкообразно за очень короткое время (практически мгновенно), иколичество этих состояний известно (фиксировано).

Под непрерывным временем будем понимать такое, при котором переход из одногодопустимого состояния в другое допустимое состояние происходит в произвольныемоменты времени, т. е. заранее не определенные.

***Потоки событий.***Однородные события, следующие друг за другом в произвольныемоменты времени (случайно), называются потоком событий (или входным потокомзаявок). Примерами потоков событий могут быть: поток пассажиров в авиакассе, потокпосетителей парикмахерской, поток отказов технического устройства и т.д. Здесь подсобытием понимается факт поступления заявок на обработку (приход покупателя,наличие отказа технического средства, поступление телефонного вызова и т.д.), а не результат его обработки (как это рассматривается в теории вероятностей). Поэтому всистемах массового обслуживания вероятностными характеристиками будет обладатьне отдельное событие, а интервал времени.

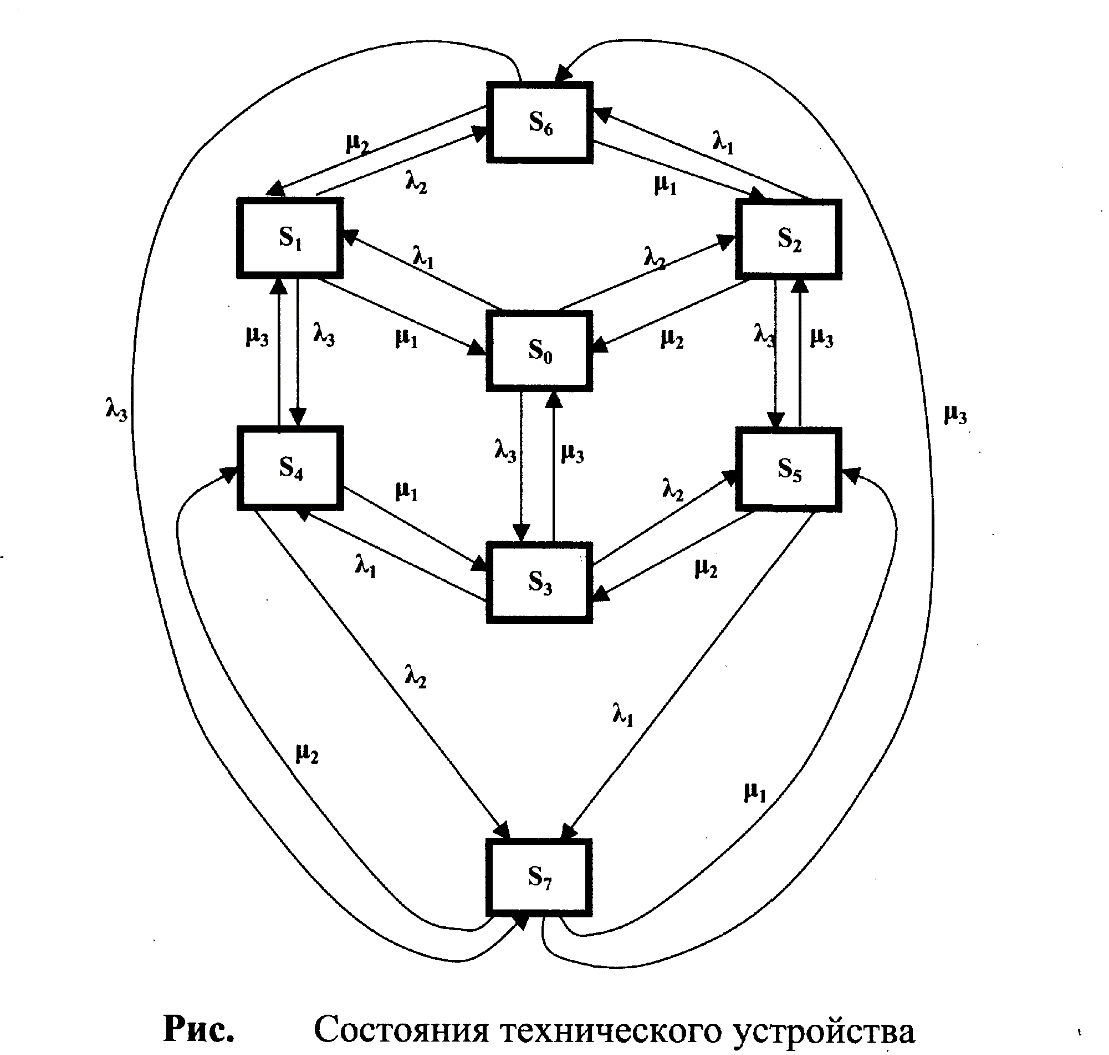
Интенсивностью потока событий называется среднее число событий за единицувремени. Интенсивность может быть как числом постоянным (константой), так ивеличиной, зависящей от времени *t.*Например, количество пассажиров в городскомтранспорте в «часы пик» резко увеличивается по сравнению с другим временем суток.



**Финальные вероятности состояний**

Будем рассматривать марковские процессы с дискретными состояниями и непрерывнымвременем.

***Пример 1***: Техническое устройство состоит из трёх узлов и в любой момент времениможет находиться в одном из восьми состояний (рис. 1).



Возможные состояния устройства таковы:

S0 — все три узла исправны;

S1— первый узел неисправен, второй и третий исправны;

S2 — второй узел неисправен, первый и третий исправны;

S3 — третий узел неисправен, первый и второй исправны;

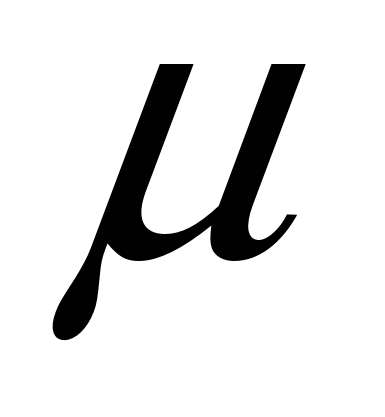
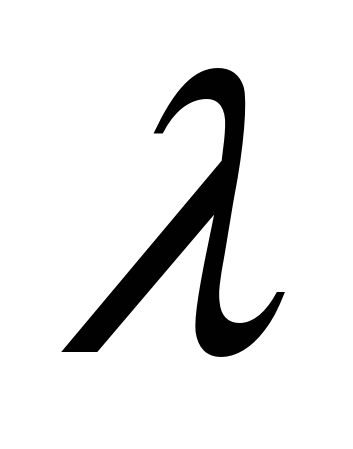
S4 — первый и третий узлы неисправны, второй исправен;

S5 — второй и третий узлы неисправны, первый исправен;

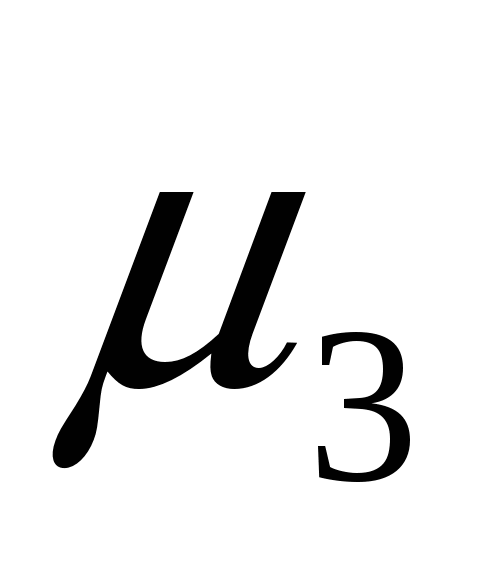
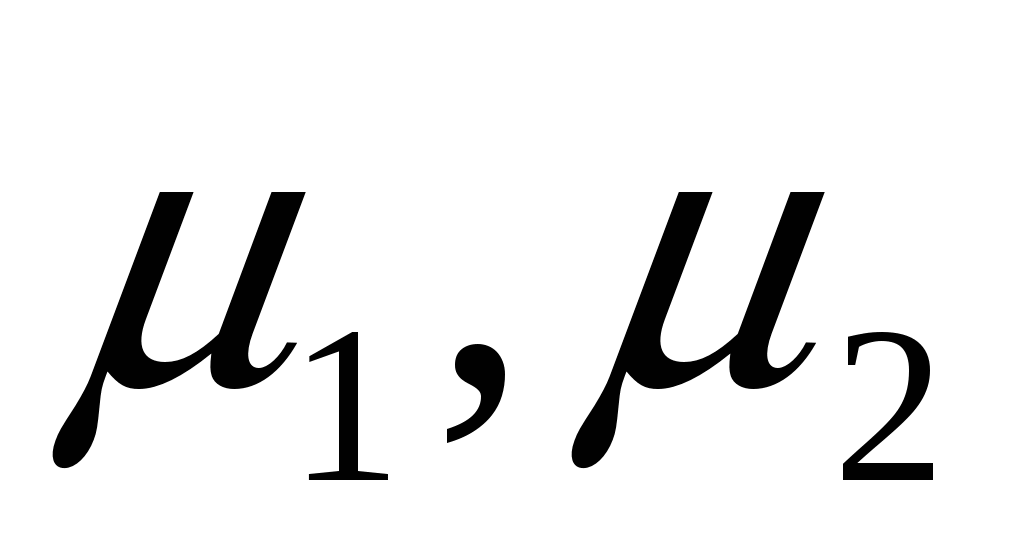
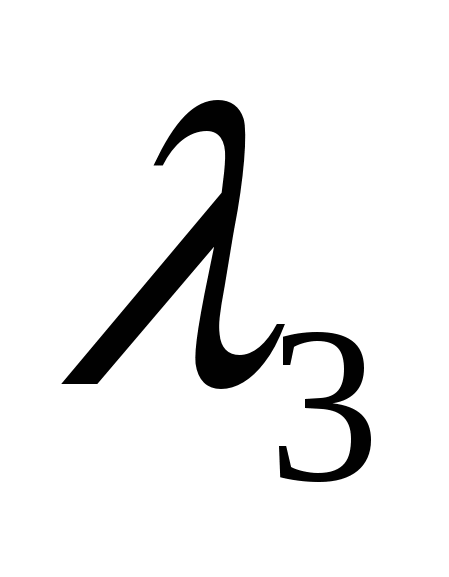
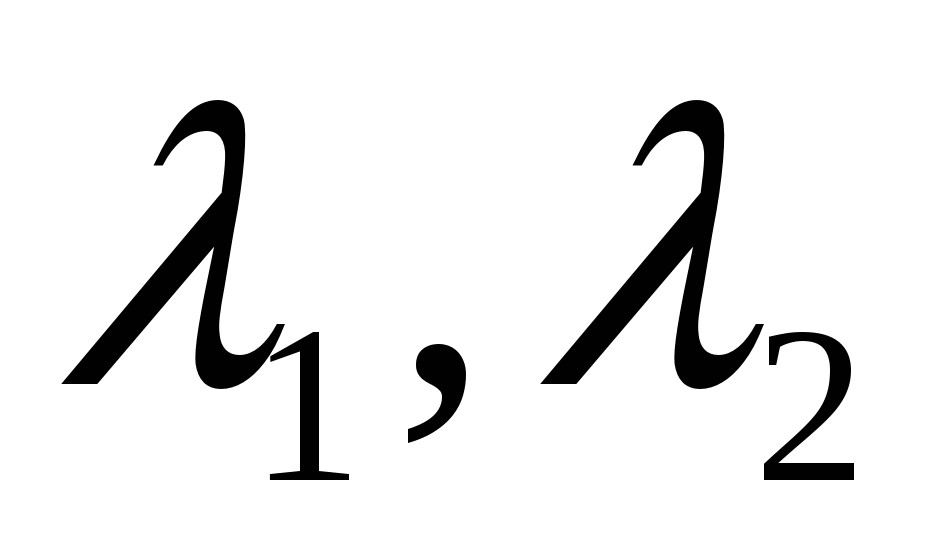
S6 — первый и второй узлы неисправны, третий исправен;

S7 — все три узла неисправны.

Размеченным графом будем считать такой граф, у которого стрелками указаныпереходы из одного состояния в другое, а рядом со стрелкой указана интенсивностьперехода. Будем различать две интенсивности — прямую , и обратную .



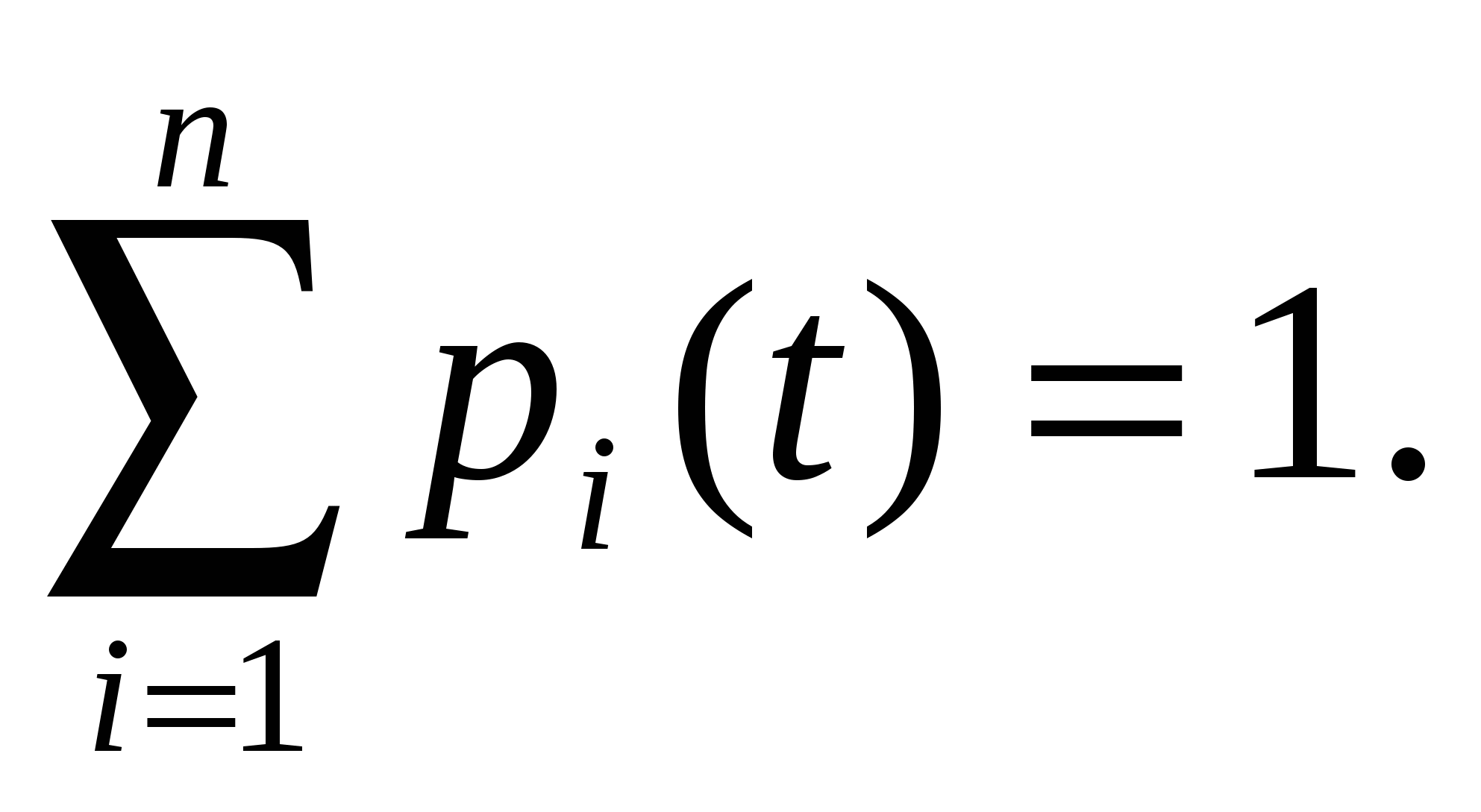
Тогда  и — интенсивности потоков отказов соответственно первого, второго итретьего узлов, а и — соответственно интенсивности потоков возвратов(ремонтов) узлов.



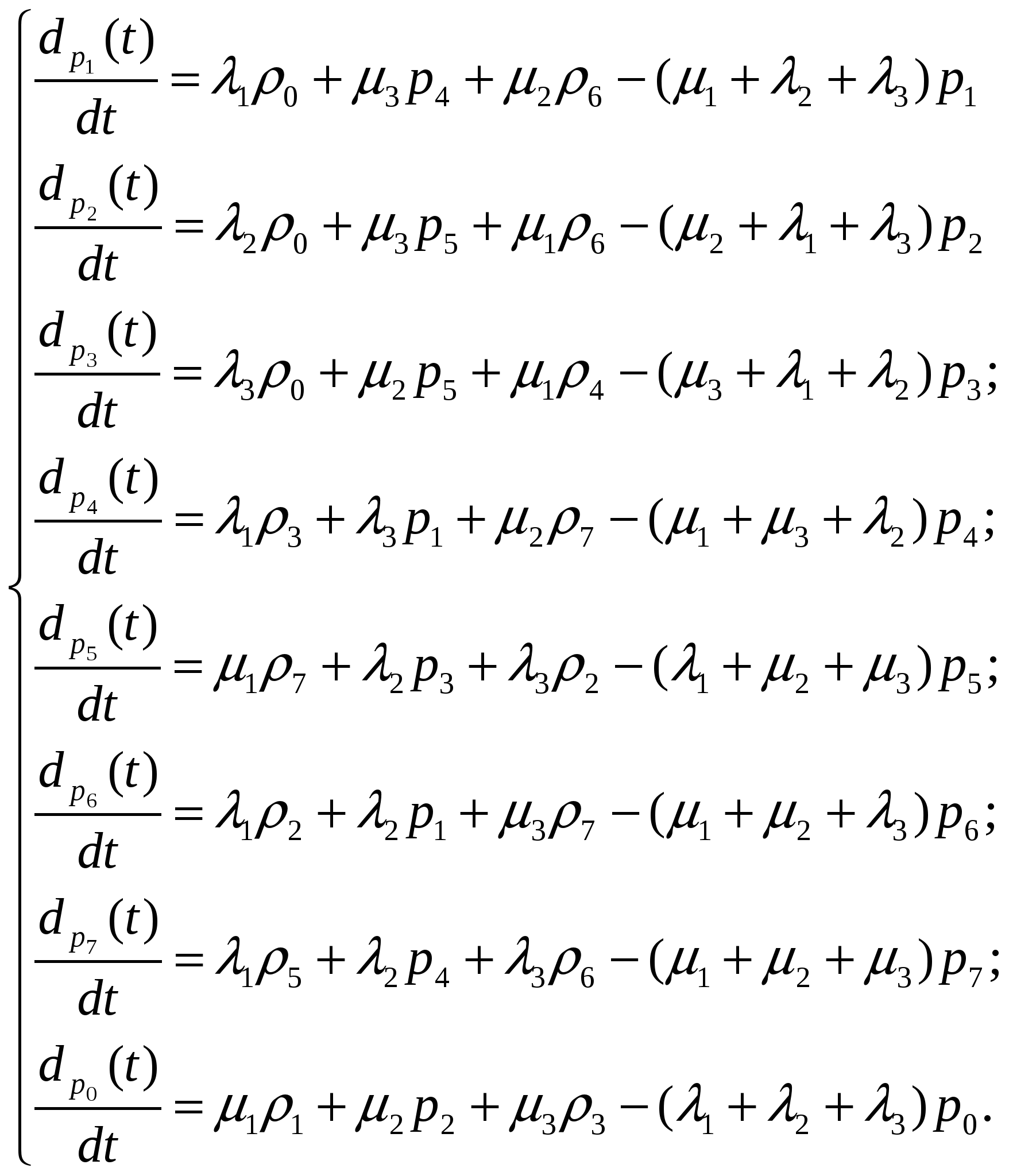
Если для ремонта каждого узла имеется отдельный специалист, то среднее времяремонта каждого узла есть величина постоянная и не имеет значения, один илинесколько узлов вышли из строя.

На основе построенного размеченного графа (см. рис. 1) создадим математическуюмодель.

Наше техническое устройство в соответствии с построенным графом в любой моментвремени будет находиться в одном из восьми возможных состояний. Обозначимвероятность каждого*i*-го состояния как p*i*(t), тогда



Для определения вероятности каждого состояния технического устройства составимсоответствующие дифференциальные уравнения:



Эта система дифференциальных уравнений называется **системой уравненийКолмогорова.** Имеем систему из восьми линейных дифференциальных уравнений свосемью неизвестными. Известно, что сумма всех вероятностей равна единице, т. е.

*p0 +pl +p2 +p3 +p4 +p5+p6+p7=1*

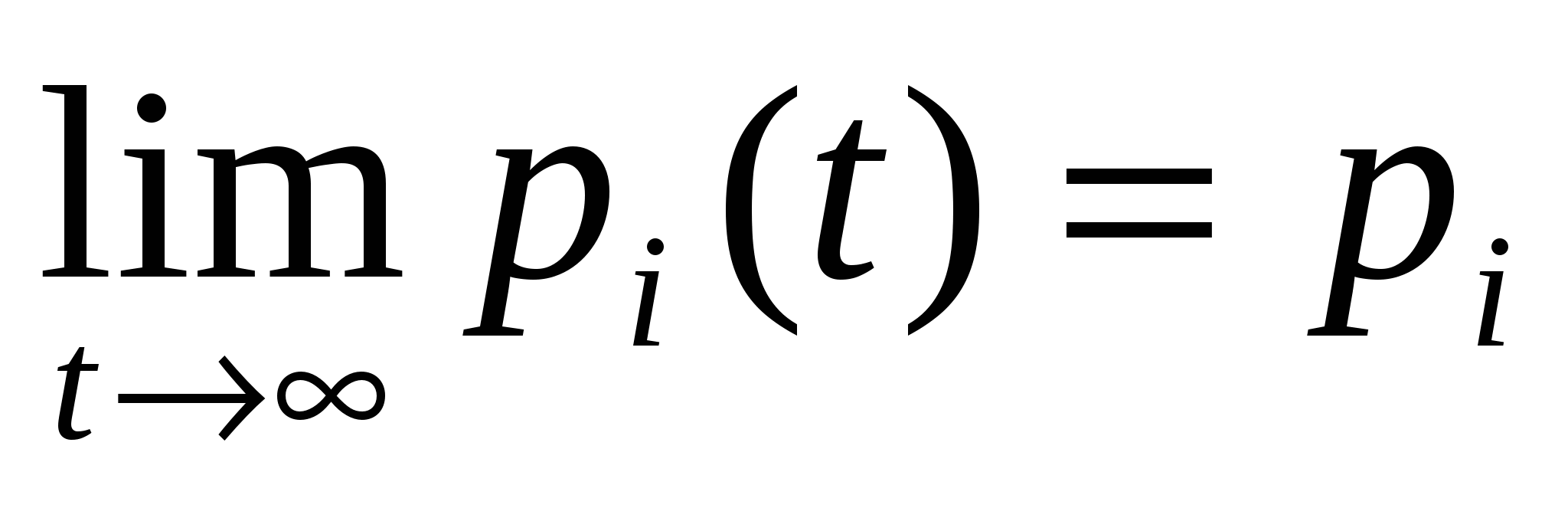
Таким образом, любое из уравнений, входящее в систему уравнений, можно записать,используя последнее уравнение, и найти значения вероятностей для каждого события.

Для облегчения процесса составления дифференциальных уравнений можно применитьследующее правило:

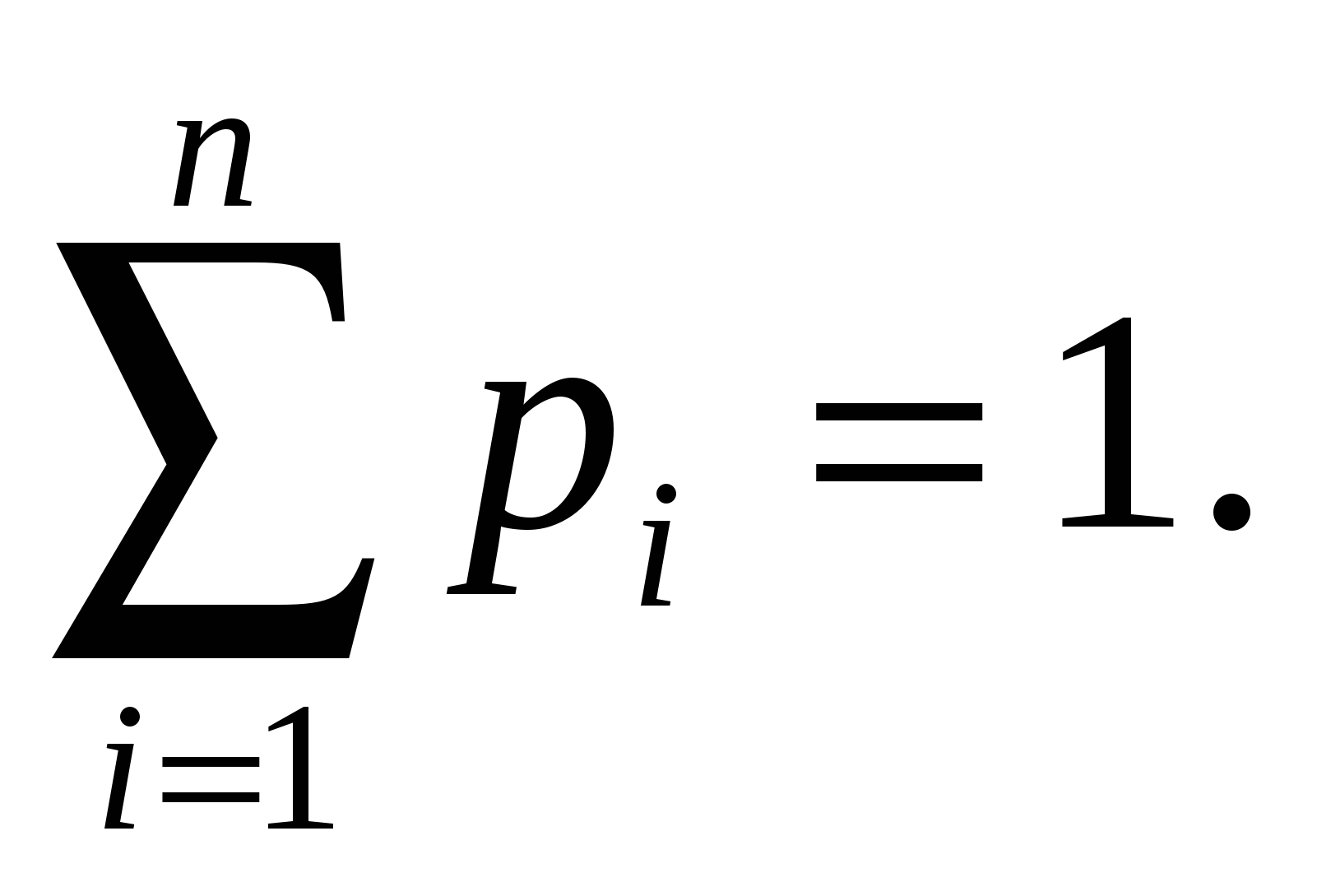
*В левой части каждого уравнения следует записать производную вероятности г-госостояния устройства.*

*В правой части сумма произведений потока событий, входящих в текущее состояние,умноженная на вероятность состояния, из которого исходит поток, минус суммарнаяинтенсивность исходящих потоков событий из текущего состояния, умноженная навероятность текущего состояния.*

Если финальные вероятности существуют:  при *i = 1, 2, 3, ..., n,*



то их сумма будет равна единице:



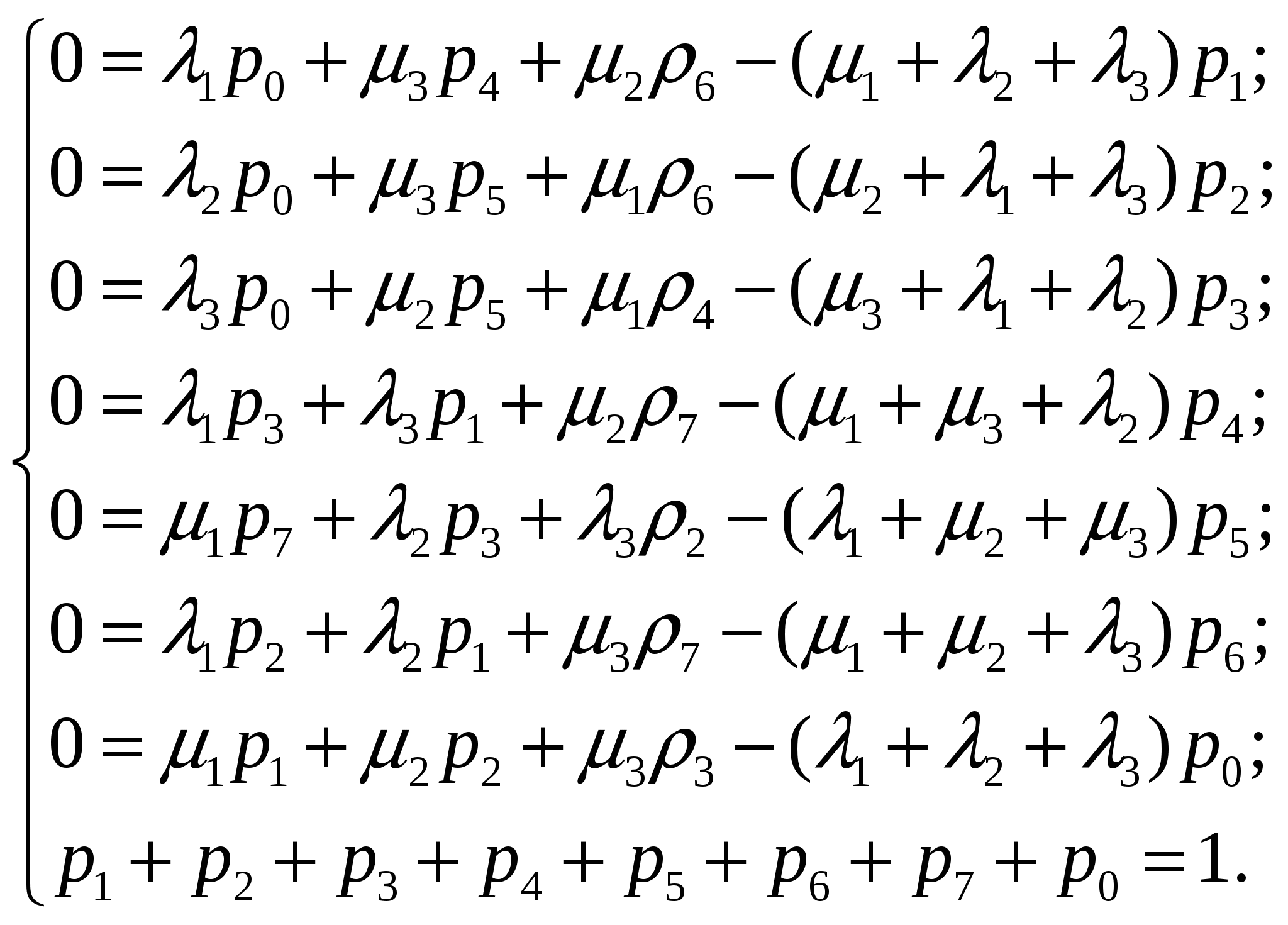
Финальные вероятности показывают, какое среднее время устройство будет находитьсяв каждом состоянии. Финальные вероятности находятся из системы дифференциальныхуравнений, если их правые части приравнять нулю.

**Решение системы уравнений Колмогорова**

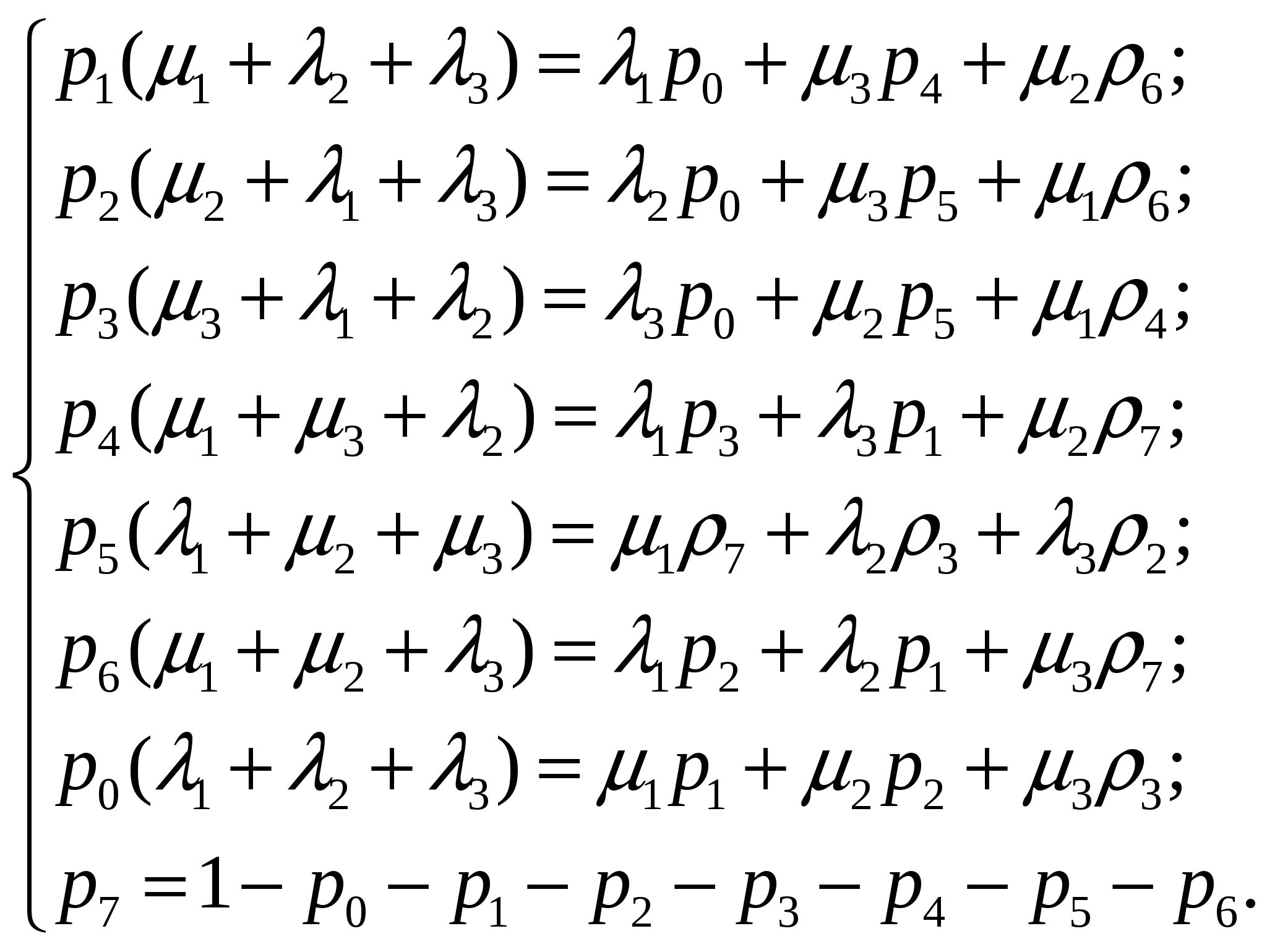
Зададим численные значения интенсивности потоков событий для примера 1:

λ1=1; λ2=2; λ3=1; μ1=2; μ2=4; μ3=2.

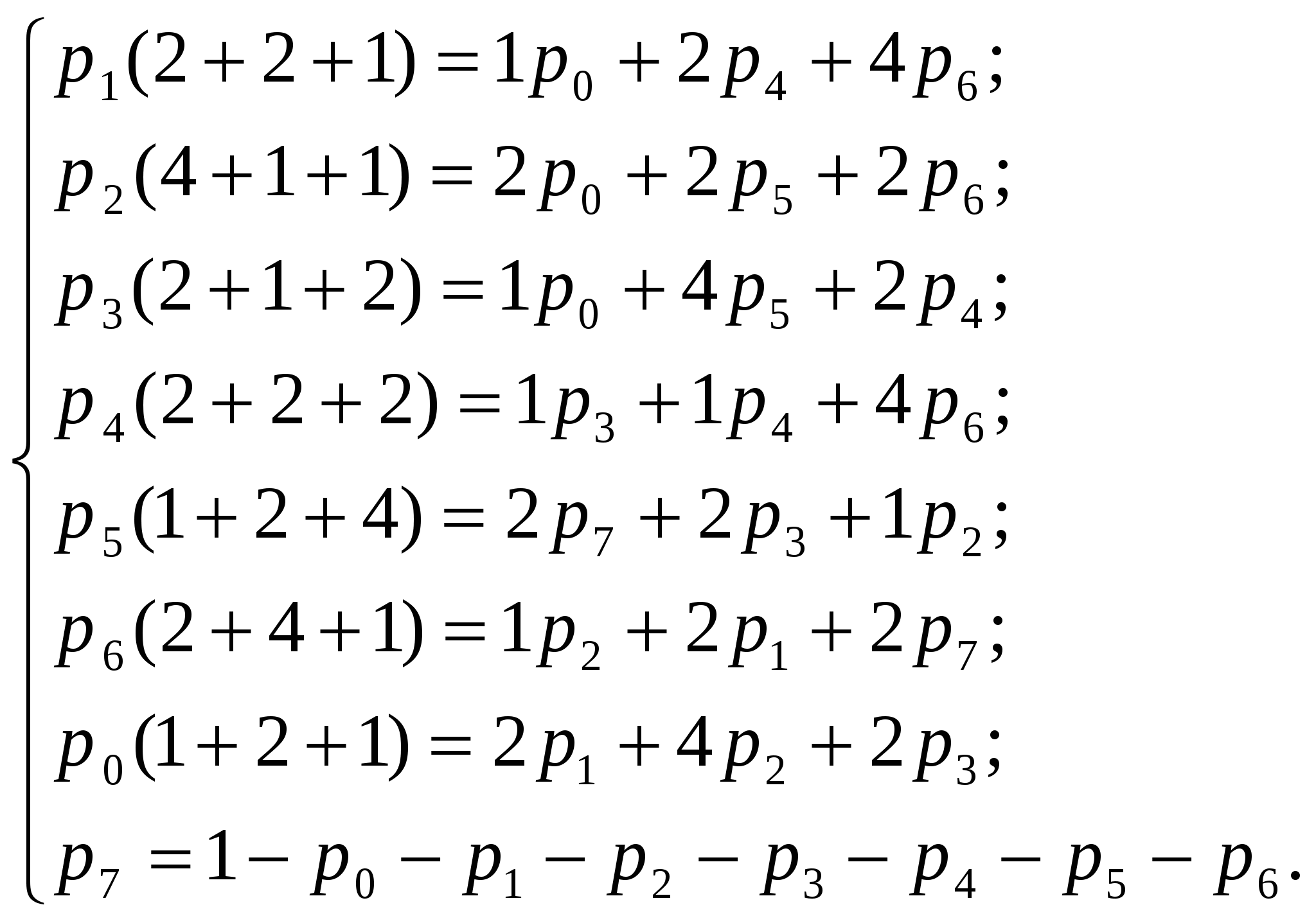
Приравняем левые части уравнений системы нулю .



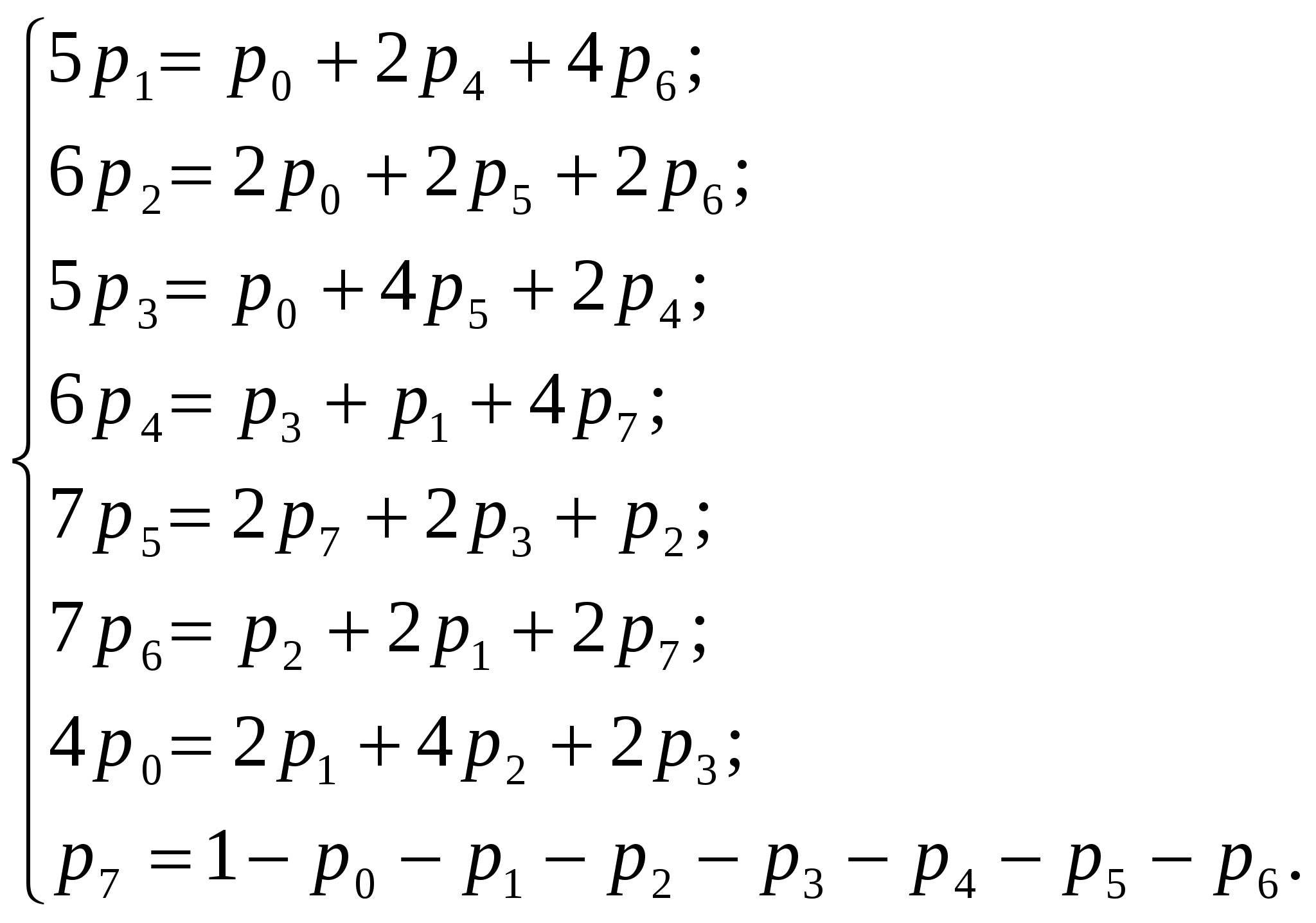
Второй (отрицательный) член каждого выражения перенесем в левую часть



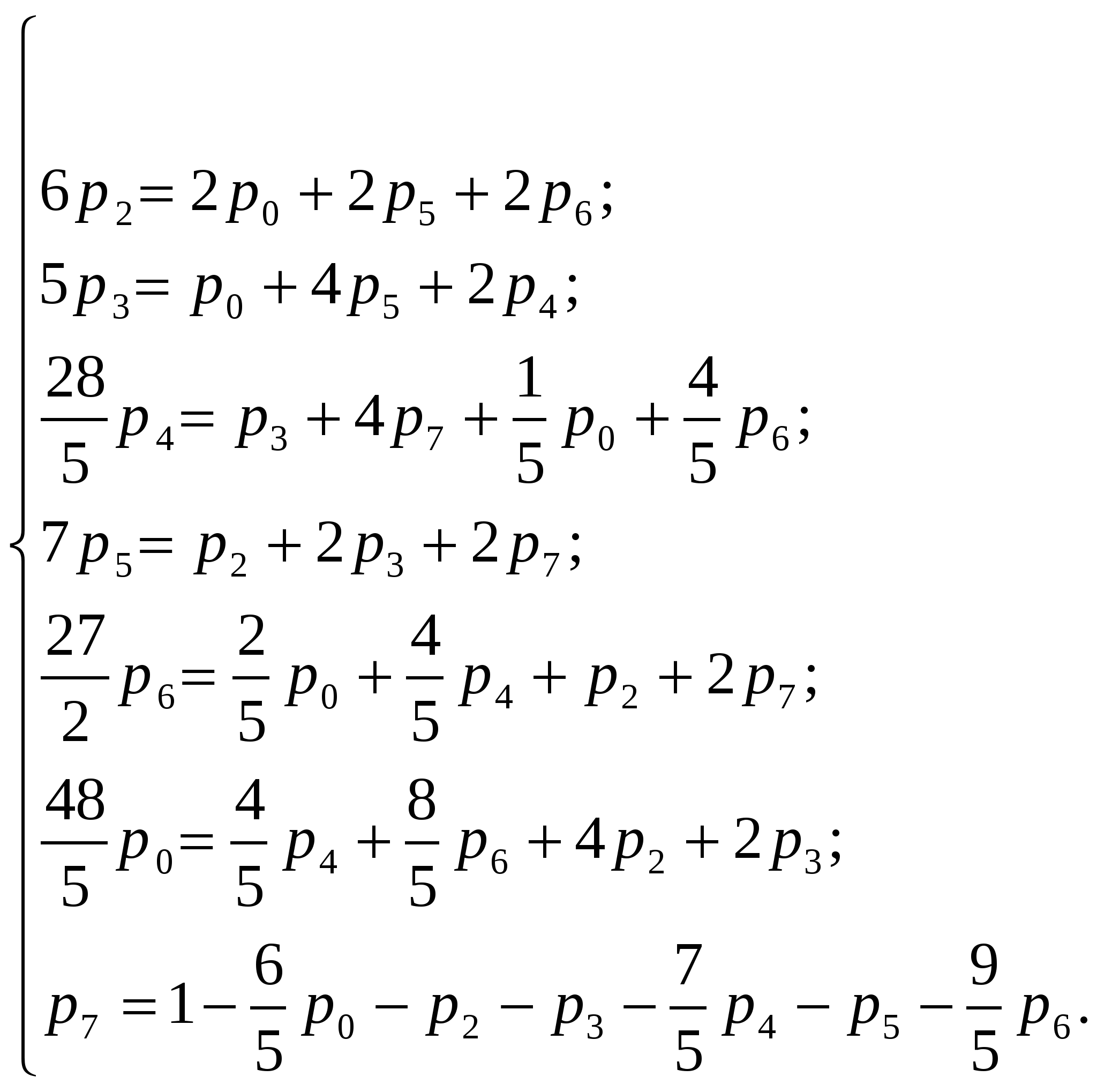
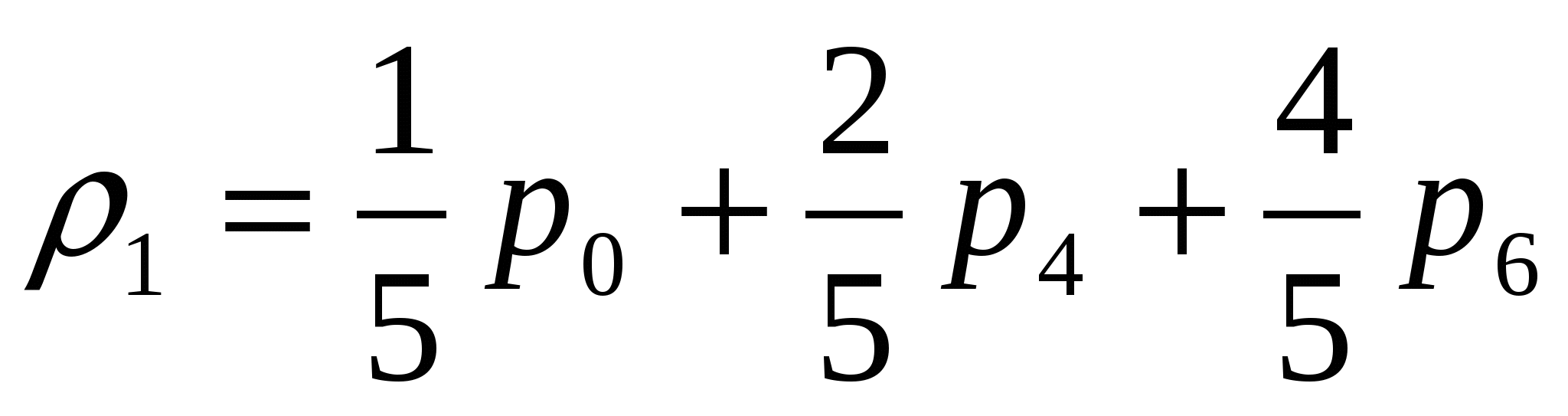
Подставим конкретные значения (указанные выше) прямых и обратных интенсивностей



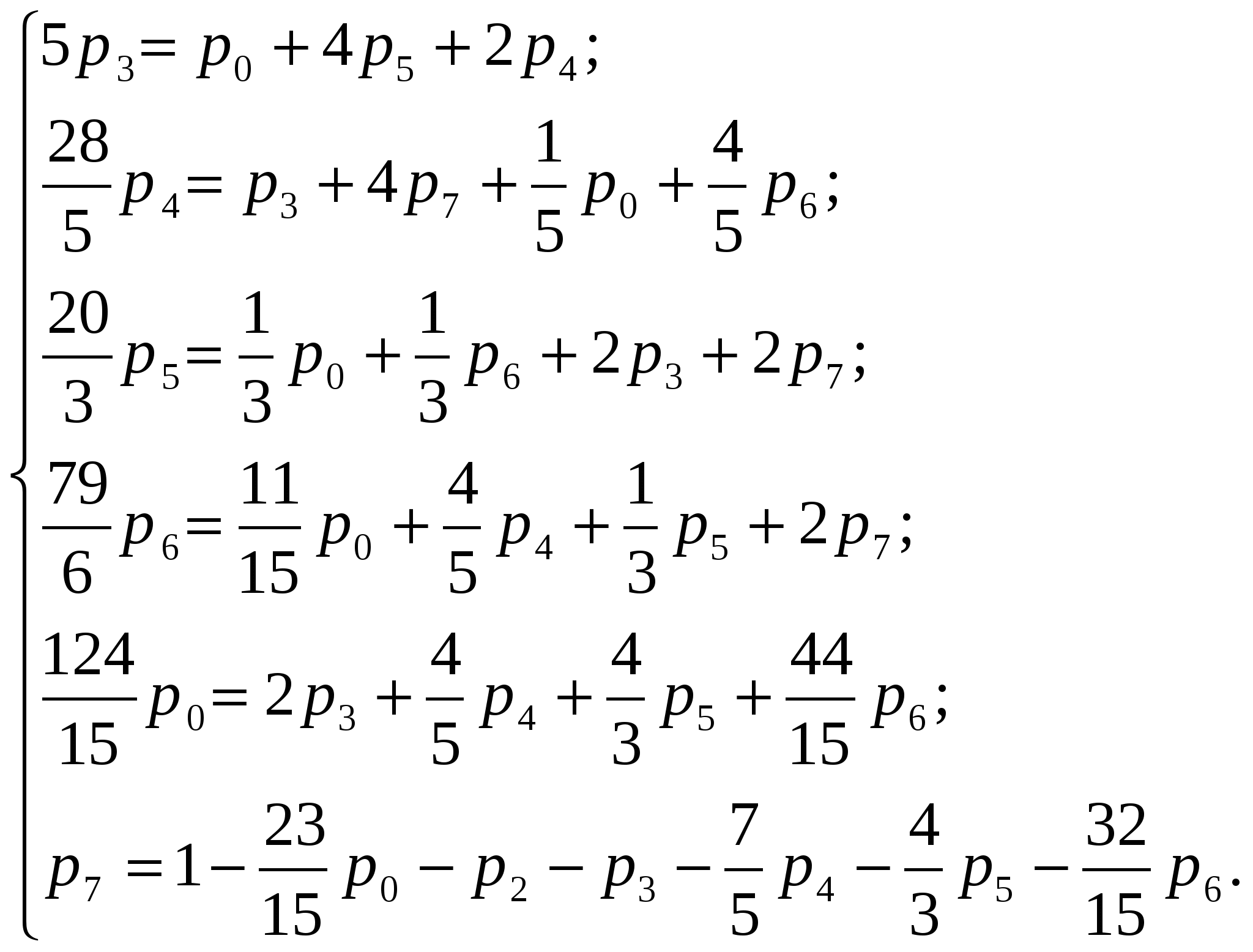
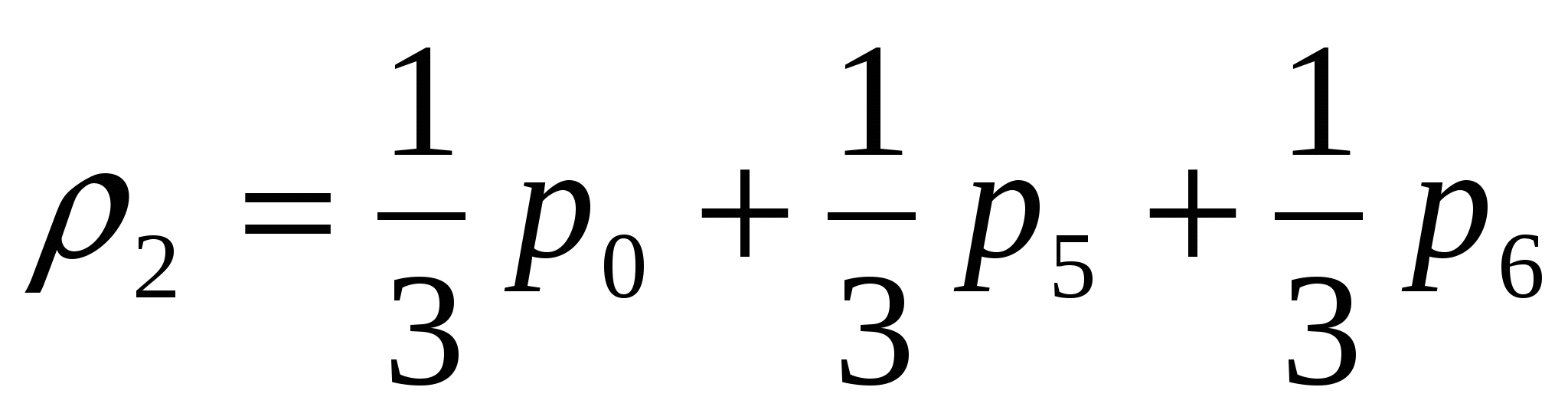
После выполнения арифметических действий получим:



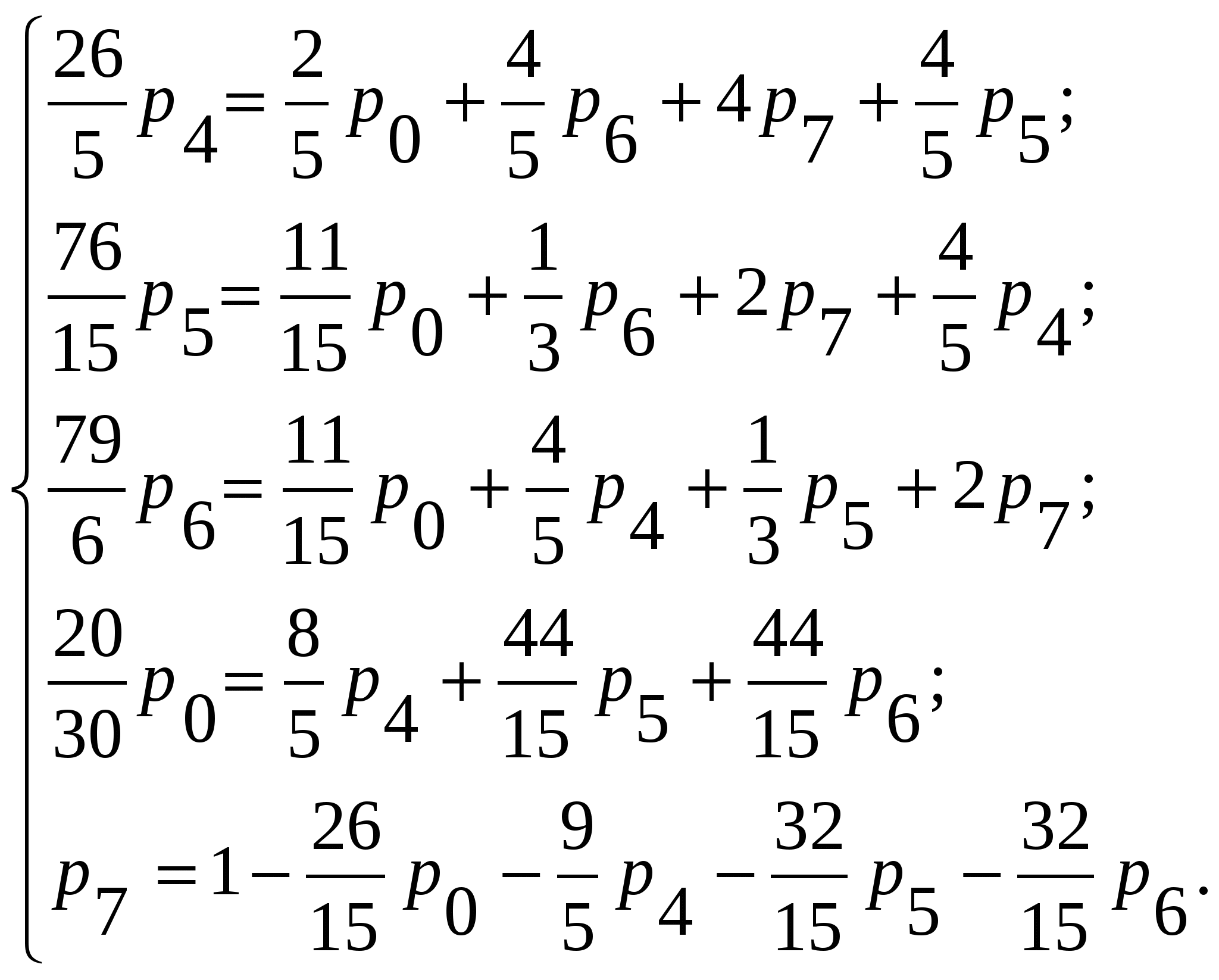
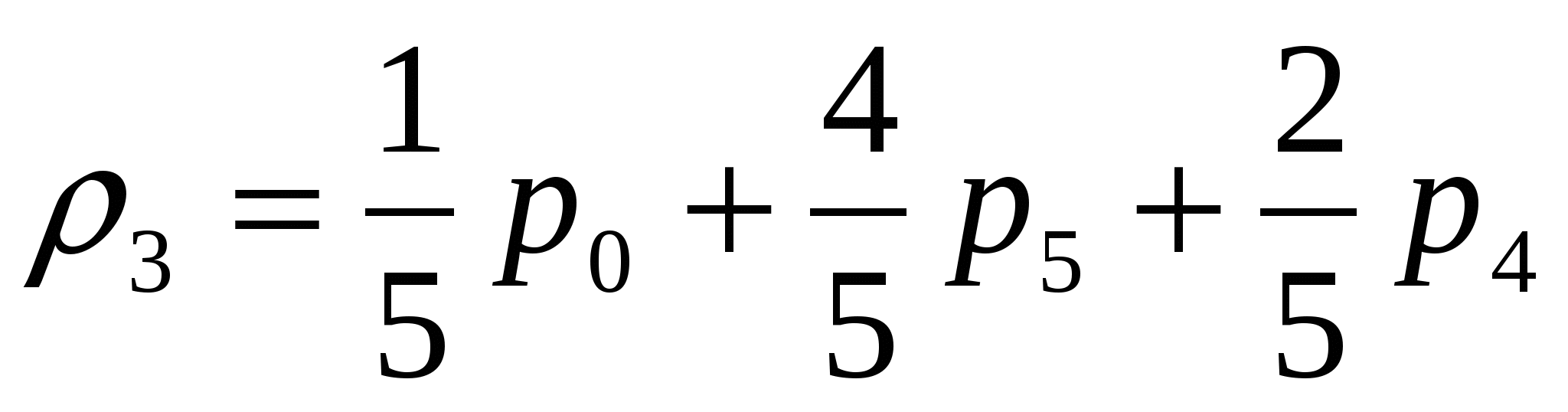
Из первого уравнения выразим  и подставим его в остальные уравнения:



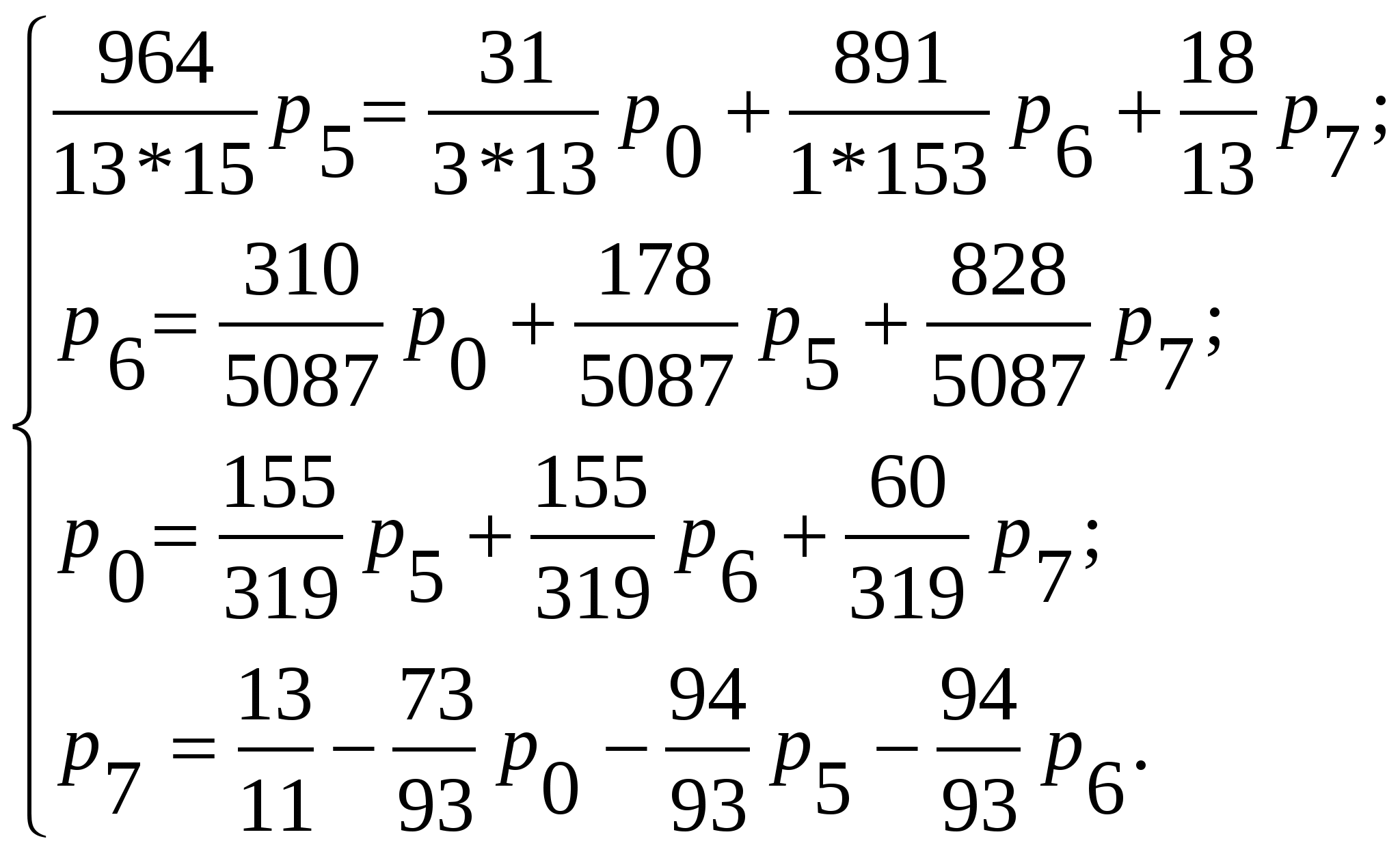
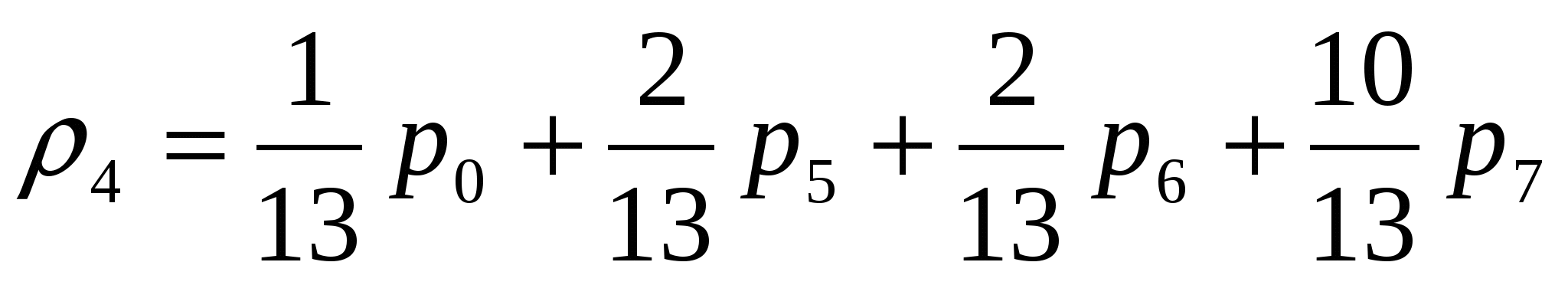
Аналогично выражаем  и подставляем в оставшиеся уравнения и получаем:



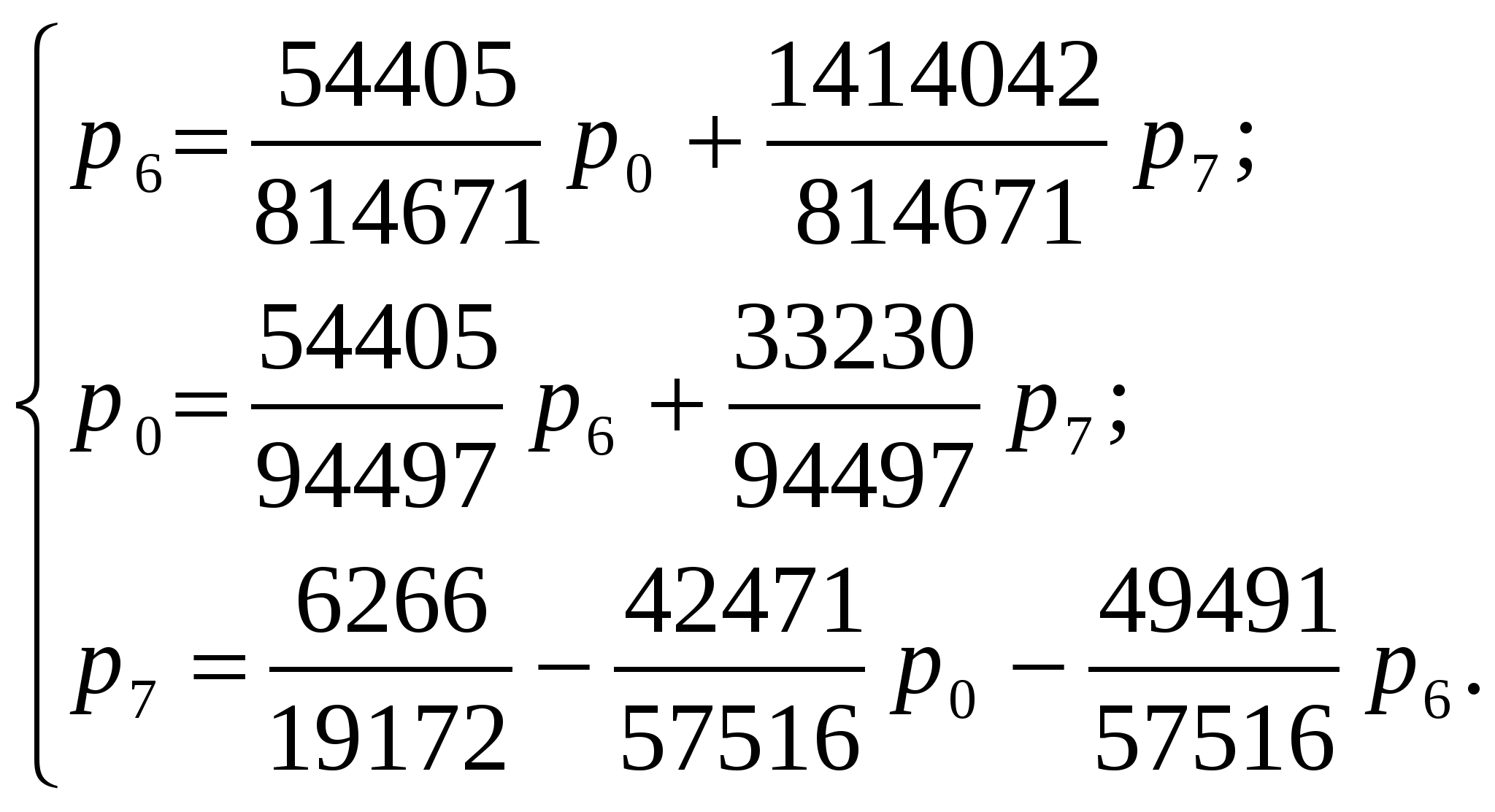
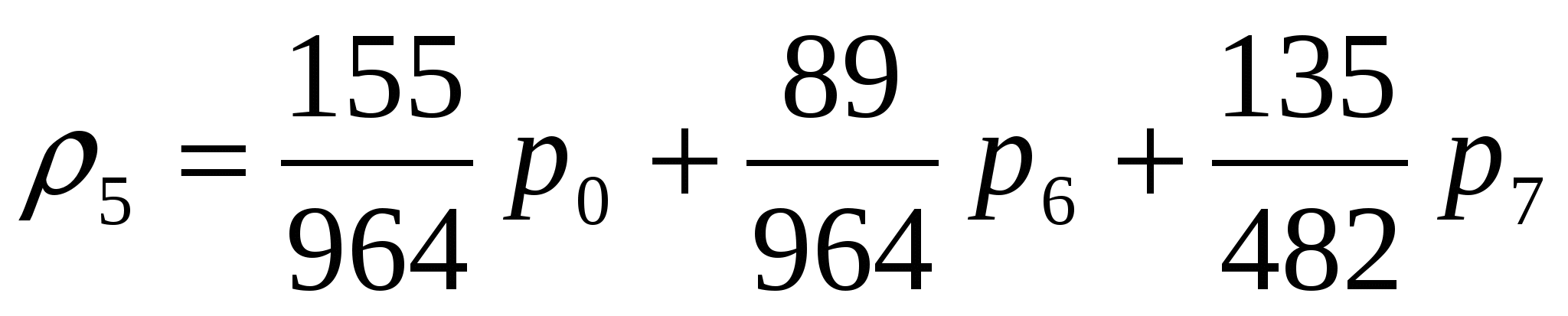
Выражаем  и подставляем в оставшиеся уравнения и получаем:



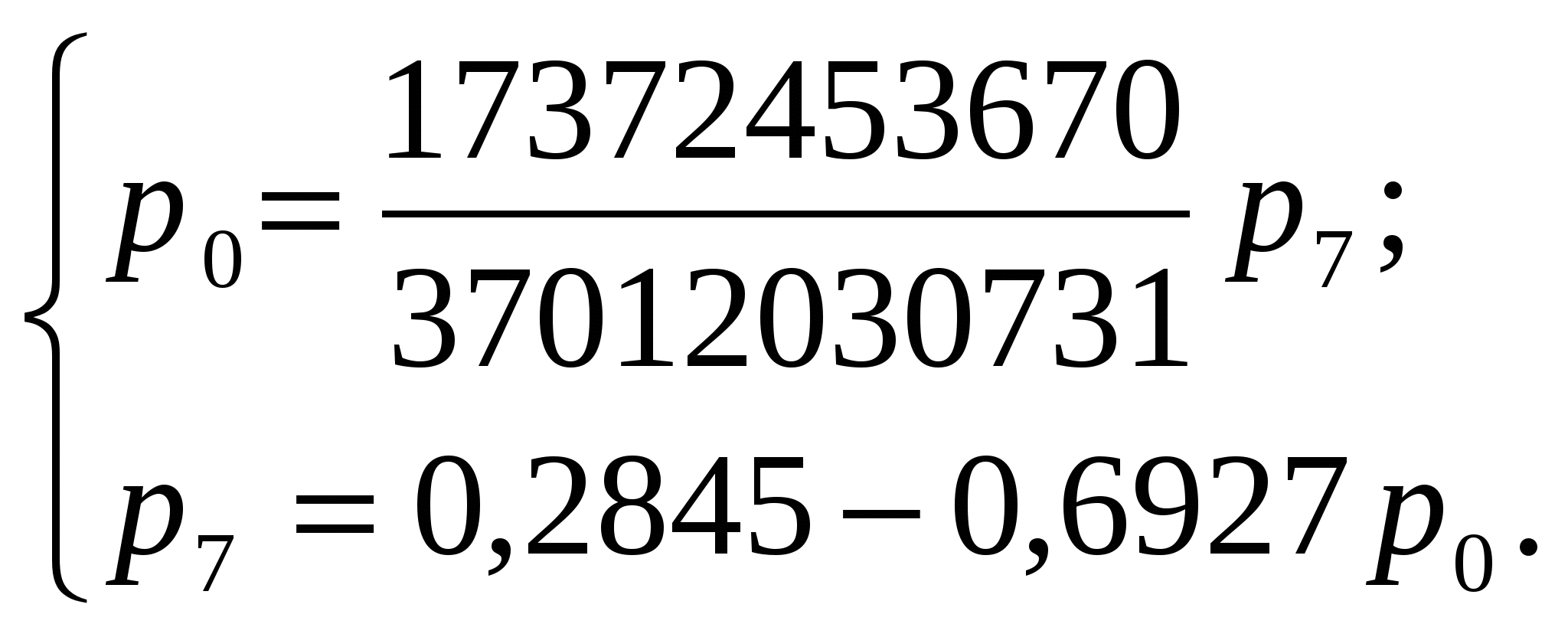
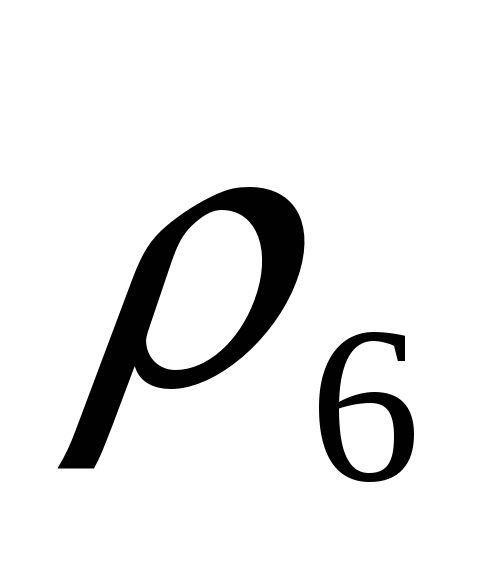
Из первого выражения выразим  и подставим в оставшиеся уравнения. После выполнения преобразований получим:



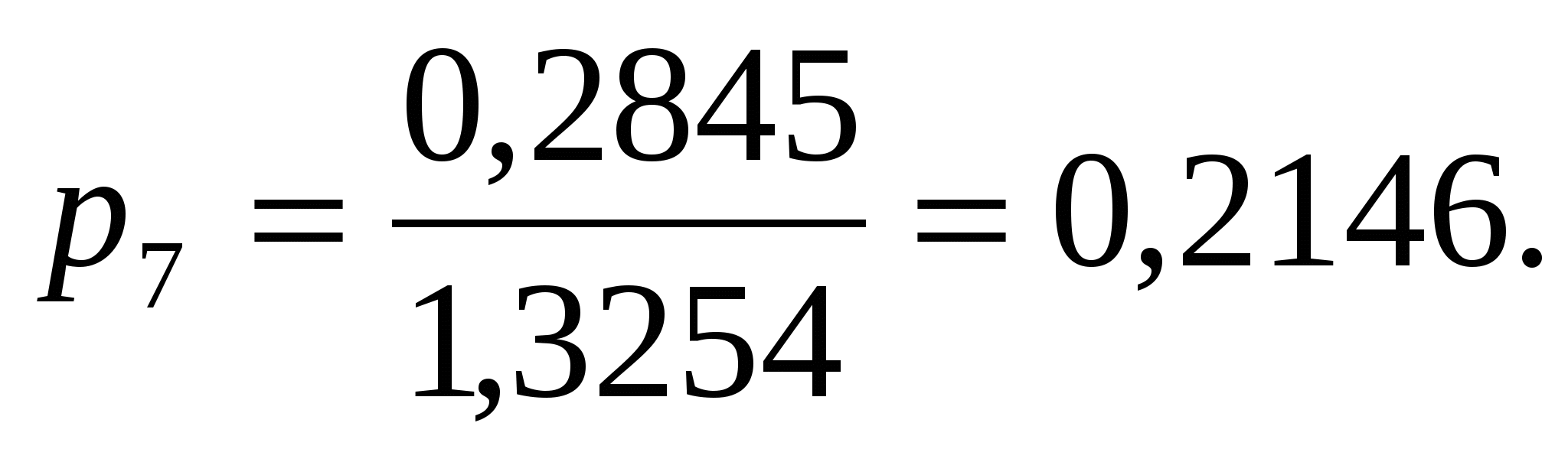
Из первого уравнения выразим и подставим в оставшиеся уравнения:



Из первого уравнения в оставшиеся уравнения:



Из первого уравнения *p0* подставим в оставшиеся уравнения:



Определим остальные вероятности, подставляя полученные результаты в обратном порядке

*P0*= 0,46940 \*0,21146=0,1007;

*P6=*0,06678 \*0,107 +0,1731 \*0,2146=0,04387;

*P5*= 0,1608\*0,1007+0,09232\*0,04387+0,2801\*0,2146=0,08035;

*P4*=0,07692\*0,1007+0,1538\*0,08035+0,1538\*0,04387+0,7692\*0,2146=0,08035;

*P3*=0,2\*0,1007+0,8\*0,080035+0,4\*0,1853=0,1585;

*P2*=0,3333\*0,1007+0,3333\*0,08035+0,3333\*0,04387=0,07498;

*P1****=***0,2\*0,1007+0,4\*0,1853+0,8\*0,04387=0,1294.

Выполним проверку. Сумма вероятностей всех событий должна быть равна единице.

*p0+p1+p2+p3+p4+p5+p6+p7*=1

0,1294+0,07498+0,1585+0,1853+0,08035+0,043870+0,04387+0,1007+0,2146=0,9877

Полученный результат меньше единицы, так как значение каждой вероятности было округленно.

**Практическая работа №33 «Решение задач массового обслуживания методами имитационного моделирования»**

**Цель работы**: ознакомиться с имитационным моделированием в задачах поиска управленческих решений. Изучить основные понятия и разновидности имитационного моделирования.

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

**Задание и порядок выполнения работы**

1. Ознакомиться с теоретическими положениями, приведёнными в работе.
2. Описать имитационные модели.
3. Изучить метод Монте-Карло.
4. Сделать Вывод.

Основы теории

1.1 Имитационные модели. Основные понятия.

Разновидности имитационного моделирования

Одним из основных направлений, определяющих методоло­гию, концептуальные и реализационные основы информационной технологии поддержки принятия управленческих решений, является имитационное моделирование. Методология имитационного моде­лирования основана на воспроизведении реальных или гипотетиче­ских бизнес-процессов в специальной компьютерной среде, обра­зующей виртуальный мир предприятия, организации, производства и любого другого объекта управления.

Эта технология появилась в 60-х г. XX в., и на протяжении многих лет она не только остается одной из основных в исследова­нии операций, но и бурно развивается в области реинжиниринга бизнес-процессов и новых направлений искусственного интеллекта.

Необходимость в построении и использовании имитационной модели возникает в том случае, если объект моделирования на­столько сложен, что описать его поведение традиционными метода­ми сложно или невозможно. Во многих случаях имитационный :жс- перимеш является единственным способом описания реальных сложных систем, поскольку процессы, протекающие в таких систе­мах, являясь многокритериальными, связаны, кроме того, с дли­тельными временными интервалами.

Основу такой технологии составляет компьютерный имита­ционный эксперимент, связанный с воспроизведением динамиче­ских процессов функционирования исследуемой системы. В процес­се воспроизведения реальной системы с помощью имитационной модели осуществляется наблюдение за функционированием модели и выявление «узких мест» в организации деятельности. Основными достоинствами лого метода являются:

* возможность воспроизведения реальной системы с практи­чески любым уровнем детальности;
* повторяемость эксперимента;
* возможность произвольной фрагментации и структуризации системы.

*Имитационное моделирование* (от англ. simulation) - это распространенпая разновидность аналогового моделирования, реали­зуемого с помощью набора математических инструментальных средств, специачьных имитирующих компьютерных программ и технологий программирования, позволяющих посредством процес- сов-аналогов провести целенаправленное исследование структуры и функций реального сложного процесса в памяти компьютера в ре­жиме «имитации», выполнить оптимизацию некоторых его пара­метров.

*Имитационной моделью* называется специальный программ­ный комплекс, который позволяет имитировать деятельность како­го-либо сложного объекта. Он запускает в компьютере параллель­ные взаимодействующие вычислительные процессы, являющиеся по своим временным параметрам (с точностью до масштабов вре­мени и пространства) аналогами исследуемых процессов. В странах, занимающих лидирующее положение в создании новых компью­терных систем и технологий, научное направление Computer Science использует именно такую трактовку имитационного моделирования.

Следует отметить, что любое моделирование имеет в своей методологической основе элементы имитации реальности с помо­щью какой-либо символики (математики) или аналогов. Поэтому иногда имитационным моделированием стали называть целенаправ­ленные серии многовариантных расчетов, выполняемых на компью­тере с применением экономико-математических моделей и методов. Однако с точки зрения компьютерных технологий такое моделиро­вание (modelling) - это обычные вычисления, выполняемые с помо­щью расчетных программ или табличного процессора Eхсеl.

Математические расчеты (в том числе табличные) можно производить и без ЭВМ: используя калькулятор, логарифмическую линейку, правила арифметических действий и вспомогательные таблицы. Но имитационное моделирование - это чисто компьютер­ная работа которую невозможно выполнить подручными средства­ми. Поэтому часто для этого вида моделирования используется си­ноним*компьютерное моделирование.*

Имитационную модель нужно создавать. Для этого необходи­мо специальное программное обеспечение -*система моделирования*(simulation system). Специфика такой системы определяется техно­логией работы, набором языковых средств, сервисных программ и приемов моделирования.

Имитационная модель должна отражать большое число пара­метров, логику и закономерности поведения моделируемого объекта во времени*(временная динамика)* и в пространстве*(пространст­венная динамика).*Моделирование сложных объектов, в частности, в области автосервиса и автоперевозок, связано с понятием*финан­совой динамики* объекта.

В общем случае, с точки зрения специалиста в области ин­формационных технологий,*имитационное моделирование* кон тро­лируемого процесса или управляемого объекта - это высокоуровне­вая информационная технология, которая обеспечивает два вида действий, выполняемых с помощью компьютера:

1)работы по созданию или модификации имитационной мо­дели;

2)эксплуатацию имитационной модели и интерпретацию ре­зультатов.

Имитационное моделирование реализуется посредством набо­ра математических инструментальных средств, специальных ком­пьютерных программ и приемов, позволяющих с помощью компью­тера провести целенаправленное моделирование в режиме «имита­ции» структуры и функций сложного процесса и оптимизацию не­которых его параметров. Набор программных средств и приемов моделирования определяв! специфику системы моделирования - специального программного обеспечения.

В отличие от других видов и способов математического моде­лирования с применением ЭВМ имитационное моделирование име­ет свою специфику: запуск в компьютере взаимодействующих вы­числительных процессов, которые являются но своим временным параметрам - с точностью до масштабов времени и пространства - аналогами исследуемых процессов.

Имитационное моделирование как особая информационная технология состоит из следующих основных этапов:

*Структурный анализ процессов.* Проводится формализация структуры сложного реального процесса путем разложения его на подпроцессы, выполняющие определенные функции и имеющие взаимные функциональные связи согласно легенде, разработанной рабочей экспертной группой. Выявленные подпроцессы, в свою очередь, могут разделяться на другие функциональные подпроцес­сы. Структура общего моделируемого процесса может быть пред­ставлена в виде графа, имеющего иерархическую многослойную структуру. В результате появляется формализованное изображение ими гационой модели вграфическом виде.

Структурный анализ особенно эффективен при моделирова­нии сложных организационных и управленческих процессов, где (в отличие от простых технических систем, основанных на физиче­ском моделировании) многие составляющие подпроцессы не имеют физической основы и протекают виртуально, поскольку оперируют с информацией, деньгами и лог икой (законами) их обработки.

*Формализованное описание модели.* Графическое изображение имитационной модели, функции, выполняемые каждым подпроцес­сом, условия взаимодействия всех подпроцессов и особенности по­ведения моделируемого процесса (временная, пространственная и финансовая динамика) должны быть описаны на специальном языке для последующей трансляции. Для этого существуют различные способы:

Описание вручную на языке типа GPSS, Pilgrim и даже на Visual Basic. Последний очень прост, на нем можно запрограммиро­вать элементарные модели, но он не подходит для разработки ре­альных моделей сложных экономических процессов, так как описа­ние модели средствами Pilgrim компактнее аналогичной алгоритми­ческой модели на Visual Basic в десятки-сотни раз;

Автоматизированное описание с помощью компьютерного графического конструктора во время проведения структурного ана­лиза, т.е. с очень незначительными затратами на программирование. Такой конструктор, создающий описание модели, имеется в составе системы моделирование в Pilgrim.

*Построение модели (build).* Обычно это трансляция и редак­тирование связей (сборка модели), верификация (калибровка) пара­метров.

Трансляция осуществляется в различных режимах:

* интерпретации, характерном для систем типа GPSS, SLAM-II, ReThink;

- компиляции (характерен для системы Pilgrim).

Каждый режим имеет свои особенности.

Режим*интерпретации* проще в реализации. Специальная универсальная программа-интерпретатор па основании формализо­ванного описания модели запускает все имитирующие подпрограм­мы. Данный режим не приводит к получению отдельной модели­рующей программы, которую можно было бы передать или продать заказчику (продавать пришлось бы и модель, и систему моделиро­вания, что не всегда возможно).

Режим*компиляции* сложнее реализуется при создании моде­лирующей системы. Однако это не усложняет процесс разработки модели. В результате можно получить отдельную моделирующую программу, которая работает независимо от системы моделирования в виде отдельного программного продукта.

*Верификация* (калибровка) параметров модели выполняется в соответствии с легендой, на основании которой построена модель, с помощью специально выбранных тестовых примеров.

*Проведение экстремального эксперимента* для оптимизации определенных параметров реального процесса.

Системы MicroSaint, на основе использования которой рас­смотрим технологию имитационного моделирования, является од­ной из наиболее простых и адекватных задачам исследования сис­тем массового обслуживания. Система не русифицирована. Отличи­тельными особенностями ее являются концептуальная завершен­ность, целостность и простота использования даже для исследова­ния весьма сложных систем. Данная система является прекрасным инструментом для изучения концепций и технологии имитационно­го моделирования систем массового обслуживания в различных об­ластях экономики и сервиса.

*1.2. Метод МОНТЕ-КАРЛО и проверка статистических гипотез*

Статистические испытания по методу Монте-Карло представ­ляют собой простейшее имитационное моделирование при полном отсутствии каких-либо правил поведения. Получение выборок по методу Монте-Карло - основной принцип компьютерного модели­рования систем, содержащих стохастические или вероятностные элементы. Зарождение метода связано с работой фон Неймана и Улана в конце 1940-х гг., когда они ввели для него название «Мон­те-Карло» и применили его к решению некоторых задач экраниро­вания ядерных излучений. Этот математический метод был известен и ранее, но свое второе рождение нашел в Лос-Аламосе в закрытых работах по ядерной технике, которые велись под кодовым обозначением «Монте-Карло». Применение метода оказалось настолько успешным, что он получил распространение и в других областях, в частности в моделировании организационных процессов в автосер­висе и автоперевозках.

Поэтому многим специалистам термин «метод Монте-Карло» иногда представляется синонимом термина «имитационное модели­рование», что в общем случае неверно. Имитационное моделирова­ние - это более широкое понятие, и метод Монте-Карло является важным, но далеко не единственным методическим компонентом имитационного моделирования.

Согласно методу Монте-Карло, проектировщик может моде­лировать работу тысячи сложных систем, управляющих тысячами разновидностей подобных процессов, и исследовать поведение всей группы, обрабатывая статистические данные. Другой способ приме­нения этого метода заключается в том, чтобы моделировать поведе­ние системы управления на очень большом промежутке модельного времени (несколько лет), причем астрономическое время выполне­ния моделирующей программы па компьютере может составить до­ли секунды.

Применение надстроек Ехсе1 для решения задач, связанных с имитационными моделями, в основном базируется на данном мето­де, поскольку он является основой получения модельных выборок по заданному закону распределения.

Рассмотрим метод Монте-Карло подробнее.

В различных задачах, встречающихся при создании сложных систем, могут использоваться величины, значения которых опреде­ляются'случайным образом. Примерами таких величин являются:

- случайные моменты времени, в которые поступают заказы на фирму;

- загрузка производственных участков или служб объектов ав­тосервиса;

- внешние воздействия (требования или изменения законов, платежи по штрафам и др.);

- оплата банковских кредитов при покупке автомобиля;

- поступление средств от заказчиков предприятий автосерви­са;

-ошибки измерений при проведении испытаний новых моде­лей автомобилей.

В качестве соответствующих им переменных могут использо­ваться число, совокупность чисел, вектор или функция. Одной из разновидностей метода Монте-Карло при численном решении задач, включающих случайные переменные, является метод статистиче­ских испытаний, который заключается в моделировании случайных событий.

Метод Монте-Карло основан на статистических испытаниях и по своей природе является экстремальным, может применяться для решения полностью детерминированных задач, таких, как обраще­ние матриц, решение дифференциальных уравнений в частных про­изводных, отыскание экстремумов и численное интегрирование, поскольку при вычислениях методом Монте-Карло статистические результаты получаются путем повторяющихся испытаний. Вероят­ность того, что эти результаты отличаются от истинных не более чем на заданную величину, есть функция количества испытаний.

В основе вычислений по методу Монте-Карло лежит случай­ный выбор чисел из заданного вероятностного распределения. При практических вычислениях эти числа берут из таблиц или получают путем некоторых операций, результатами которых являются псев­дослучайные числа с теми же свойствами, что и числа, получаемые путем случайной выборки. Имеется большое число вычислительных алгоритмов, которые позволяют получить длинные последователь­ности псевдослучайных чисел.

Один из наиболее простых и эффективных вычислительных методов получения последовательности равномерно распределен­ных случайных чисел*ri* с помощью, например, калькулятора или любого другого устройства, работающего в десятичной системе счисления, включает только одну операцию умножения.

Метод заключается в следующем: если*ri* =0,0040353607, то *ri\*1* = {403 53607*ri}* mod 1, где mod 1 означает операцию извлечения из результата только дробной части после десятичной точки. Как описано в различных литературных источниках, числа*ri* начинают повторяться после цикла из 50 миллионов чисел, так что *r5000000l= ri*Последовательность*ri* получается равномерно распределенной на интервале (0, I). Ниже будут рассмотрены более точные способы получения таких чисел со значительно большими периодами, а так­же пояснения, как в реальных моделях такие числа становятся прак­тически случайными.

Применение метода Монте-Карло может дать существенный эффект при моделировании развития процессов, натурное наблюде­ние процесса в нежелательных направлениях, но и оценивать гипо­тезы о параметрах нежелательных ситуаций, к которым приведет такое развитие, в том числе и параметрах рисков.

Существуют различные методы проверки статистических ги­потез. Наиболее широко используются на практике критерии:

* согласия / (хи-квадрат);
* Крамера-фон Мизеса;
* Колмогорова-Смиpнова.

*Критерий X* предпочтителен, если объемы выборок*N.* в от­ношении которых проводится анализ, велики. Это мощное средство, если N>*100* значений. Однако при анализе экономических ситуаций иногда бывает довольно трудно (или невозможно) найти 100 одина­ковых процессов, развивающихся с различными исходными данны­ми. Сложность заключается не только в том, что не бывает одинако­вых объектов экономики: даже если такие объекты имеются, то к исходным данным относятся не только исходные вероятностные данные и особенности структуры объекта, но и сценарий развития процессов в этом объекте и в тех объектах внешней среды, с кото­рыми он взаимодействует (процессы рынка, указы правительства, принятие новых законов, требования налоговых органов, платежи в бюджеты различных уровней). При относительно малых объемах выборок этот критерий вообще неприменим.

*Критерии Крамера-фон Мизеса* дает хорошие результаты при малых объемах выборок (при*N <* 10). Однако следует отметить

два обстоятельства:

1)при А' < 10, каким бы методом ни пользоваться, вопрос о доверительной вероятности при проверке статистической гипотезы решается плохо (эта вероятность мала при значительных размерах доверительных интервалов);

2)метод Монте-Карло используется как раз для того, чтобы недостающие данные собрать с помощью специального вычисли­тельного статистического инструментария и компьютера.

Поэтому будем полагать, что реальные объемы выборок, ко­торые можно получить, находятся в пределах 10 < N< 100. Как ука­зывают многие исследователи, для указанных пределов хорошие результаты дает*критерии Колмогорова-Смирнова* Он применяет­ся в тех случаях, когда проверяемое распределение непрерывно и известны среднее значение и дисперсия проверяемой совокупности.

**Практическая работа №34 «Построение прогнозов»**

**Цель работы**: Приобретение навыков разработки прогнозов развития экономических явлений с использованием возможностей Microsoft Excel.

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

Имеются статистические данные за несколько лет о работе фирмы в одном из направлений ее деятельности. Необходимо, используя статистические методы, рассчитать прогнозное значение интересующего показателя на следующий год, т. е. на будущий период. Кроме того, необходимо проанализировать имеющиеся данные и найти закономерность их изменения во времени. Таким образом, задание сводится к следующим этапам:

1) По приведенным данным необходимо построить прогноз с использованием скользящей средней, функции роста и тенденции. Построить графики с прогнозными данными и сравнить их с фактическими.

2) Найти наиболее точную форму зависимости между статистическими данными и временем, а также определить вид этой зависимости и ее точность, используя коэффициент детерминации R2. Используя уравнение регрессии, найти значение исследуемого показателя в будущем периоде.

Указания к построению прогноза

Для примера возьмем следующие исходные данные из табл. 1

Таблица 1. Исходные данные для задачи.

|  |  |  |  |
| --- | --- | --- | --- |
| Период | Данные | Период | Данные |
| 1  2  3  4  5 | 543  555  583  628  659 | 6  7  8  9  10 | 732  843  984  1109  1000 |

Расчет прогноза с использованием скользящей средней для i-го периода производится по следующей формуле:

Для первых трех периодов рассчитать прогнозные данные нельзя, т.к. для них отсутствуют необходимые данные. Расчет начинается с четвертого периода. Для этого, после ввода исходных данных в таблицу Excel производим расчет прогноза за необходимый период (рис. 1).

Рисунок 1. Расчет значения среднего скользящего

Далее производится расчет по следующим периодам. Для быстрого расчета необходимо растянуть данную ячейку на следующие периоды, включая прогнозный. Далее необходимо сделать прогноз с использованием функции ТЕНДЕНЦИЯ. Данная функция находит линейную зависимость между заданными значениями функции y и значениями ее аргументов x. Данная зависимость представляется линейной функцией, а значения ее аргументов находятся в Excel по методу наименьших квадратов. В данном случае значения функции y — это данные, а значения аргументов x — номера периодов для соответствующих данных. Более подробную информацию о данной функции можно узнать с помощью помощника Excel, вызвав ее из меню или нажав клавишу «F1». Для расчета прогноза за необходимый период сначала введем в необходимой клетке таблицы «=ТЕНДЕНЦИЯ (», а далее укажем массив, в котором находятся известные значения функции y. Затем укажем массив, в котором находятся значения аргументов функции x. Последним указывается значение того периода, за который необходимо рассчитать прогноз, т. е. и необходимое значение x, для которого необходимо найти значение функции y.

Рисунок 2. Расчет прогноза с использованием функции тенденции

Как и для средней скользящей, данную ячейку затем можно растянуть и применить к другим периодам, включая будущий период.

Далее сделаем прогноз с использованием функции РОСТ. Данная функция Excel рассчитывает прогнозируемый экспоненциальный рост на основании имеющихся данных. Функция РОСТ возвращает значения y для последовательности новых значений x, задаваемых с помощью существующих x- и y-значений. Т. е. данная функция строит зависимость между функцией и ее аргументами в виде. О ней более подробно также можно узнать из помощника, вызвав ее из меню или нажав клавишу «F1». Для расчета прогноза за необходимый период применяется функция «РОСТ» аналогично функции «ТЕНДЕНЦИЯ» из предыдущего прогноза. Сначала необходимо указать массив, в котором находятся известные значения функции y. Затем массив, в котором находятся значения аргументов функции x, и необходимое значение x, для которого необходимо найти значение функции y.

excel экспоненциальный детерминация полиномиальный

Рисунок 3. Расчет прогноза с использованием функции роста

В итоге получаем прогноз по трем методам (рис. 4).

Рисунок 4. Расчет прогноза с использованием функции тенденции

Таким образом, можно сделать вывод, что в зависимости от используемого метода прогнозирования прогнозное значение исследуемого показателя в будущем периоде будет равняться: 1031 для средней скользящей, 1120,73 — при использовании функции тенденции, 1177,31 — при использовании функции роста. Далее, применяя «Мастер диаграмм» строим графики для данных прогнозов. Данный мастер можно вызвать из меню «Вставка» пункт меню «Диаграмма» или нажав на кнопку на панели под главным меню программы. Далее, в появившемся окне необходимо выбрать тип диаграммы. Нам необходимо выбрать тип диаграммы «График» т.к. он наиболее подходит в данном случае для отображения развития изучаемого параметра в динамике (во времени).

Рисунок 5. Построение графика прогноза по методу скользящей средней

Пример построения графика для метода средней скользящей показан на рисунке 6. На графике прогнозные значения отображаются и для будущего одиннадцатого периода, а для первых трех периодов прогнозные значения не указаны из-за специфики самого метода средней скользящей [28, "https://referat.bookap.info"].

Аналогичные графики необходимо построить для двух оставшихся прогнозов: по функции тенденции и по функции роста. Они также будут отображать прогноз на будущий период, кроме того, они будут содержать прогнозные значения и для первых трех периодов, в отличие от графика для средней скользящей.

Рисунок 6. График прогноза скользящей средней

Указания к решению второго задания

Для того, чтобы определить форму и вид зависимости между имеющимися статистическими данными и фактором времени необходимо для начала построить график, который показывает изменение изучаемого показателя во времени (в динамике).

Рисунок 7. График развития исследуемого показателя во времени

На построенном графике опробуем различные виды зависимости и определим наиболее подходящий вид. Для этого необходимо подвести курсор мышки непосредственно к линии графика, нажать правую клавишу мышки и выбрать из появившегося контекстного меню команду «Добавить линию тренда». Появится следующее окно (рис. 8), в котором необходимо подобрать наиболее оптимальный тип зависимости между изучаемыми параметрами.

Рисунок 8. Выбор типа зависимости

Также в закладке «Параметры» необходимо отметить пункты «Показывать уравнение на диаграмме» и «Поместить на диаграмму величину достоверности аппроксимации (R2)». Первый пункт необходимо отметить для того, чтобы на диаграмме отображалось уравнение регрессии, а второй — для отображения значения коэффициента детерминации R2 (рис. 9).

Рисунок 9. Изменение параметров для линии тренда

После нажатия на кнопку «ОК» добавится линия, отображающая зависимость между имеющимися данными, а также уравнение регрессии и значение R2.

Рисунок 10. Линейная зависимость между исследуемыми показателями

В данном случае был выбран линейный тип зависимости. Как видно из результатов (рис. 10), уравнение регрессии линейного вида довольно точно отображает зависимость между изучаемым параметром и временем. Значение данного коэффициента может лежать в интервале от нуля от единицы. Чем ближе значение коэффициента к R2 единице, тем точнее данное уравнение регрессии определяет зависимость между имеющимися данными, а чем ближе его значение к нулю — тем хуже данный тип зависимости подходит в данном случае. В данном случае R2 = 0,9031, т. е. линейная зависимость достаточно точно отображает зависимость изучаемого параметра от периода времени.

Однако возможно существует зависимость другого вида, которая имеет более высокое значение коэффициента детерминации. Таким образом, необходимо подобрать наиболее лучший тип зависимости. Для этого необходимо подвести курсор мышки непосредственно к линии тренда, отображающей зависимость, нажать правую клавишу мышки и выбрать из появившегося контекстного меню команду «Формат линии тренда…». В появившемся окне необходимо выбирать другие типы зависимостей и сравнивать получаемые результаты.

В данном случае наиболее точно развитие исследуемого процесса в динамике отображает полиномиальная зависимость 3й степени (рис. 11). Коэффициент детерминации наиболее близок к единице и равен 0,9555, а уравнение регрессии имеет следующий вид:

.

Подставив в данное уравнение значение будущего периода (одиннадцатого), получим прогнозное значение:

.

Рисунок 11. Полиномиальная зависимость между исследуемыми показателями

Варианты заданий (по исходным данным построить прогноз с использованием функций Скользящая Средняя, Тенденция, Рост)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Периоды | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
| Варианты |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 1 | 4,2 | 4,5 | 4,2 | 4,6 | 3,5 | 3,3 | 3,9 | 4,0 | 4,3 | 4,1 | 3,5 | 3,4 |  |  |  |  |  |  |  |
| 2 | 16,3 | 15,5 | 20,0 | 22,1 | 29,9 | 39,9 | 36,4 | 33,5 | 25,1 | 18,0 | 13,3 | 12,7 | 12,9 | 12,4 |  |  |  |  |  |
| 3 | 9,5 | 9,4 | 8,9 | 8,3 | 8,1 | 7,1 | 5,3 | 5,4 | 4,9 | 5,6 | 6,7 | 8,7 | 10,4 | 9,5 | 9,5 | 8,6 |  |  |  |
| 4 | 117 | 116 | 127 | 108 | 133 | 129 | 147 | 166 | 148 | 227 | 170 | 131 |  |  |  |  |  |  |  |
| 5 | 4272 | 4478 | 4964 | 4514 | 4430 | 3786 | 3722 | 3039 | 3029 | 2969 | 2440 | 3592 | 3517 | 3727 | 4245 |  |  |  |  |
| 6 | 33,9 | 30,3 | 23,8 | 26,1 | 26,2 | 65,9 | 43,4 | 47,4 | 43,3 | 50,3 | 35,5 | 53,7 |  |  |  |  |  |  |  |
| 7 | 7,3 | 7,4 | 7,4 | 4,2 | 3,2 | 3,3 | 2,5 | 2,5 | 5,5 | 2,3 | 14,7 | 12,8 | 13,6 |  |  |  |  |  |  |
| 8 | 302 | 257 | 256 | 281 | 296 | 267 | 320 | 324 | 293 | 278 | 289 | 300 | 318 | 268 | 288 | 312 | 282 | 308 | 281 |
| 9 | 130,0 | 129,3 | 130,8 | 131,4 | 138,1 | 142,1 | 149,4 | 145,7 | 145,2 | 147,4 | 143,8 | 155,5 |  |  |  |  |  |  |  |
| 10 | 31,0 | 31,9 | 48,6 | 37,8 | 34,4 | 44,2 | 50,8 | 42,4 | 42,7 | 37,7 | 36,3 | 42,8 | 34,3 | 36,6 | 44,6 | 42,7 | 42,1 |  |  |
| 11 | 17,8 | 16,7 | 17,7 | 17,7 | 18,5 | 16,4 | 21,4 | 21,0 | 19,5 | 18,8 | 18,5 | 18,7 | 19,6 | 22,0 |  |  |  |  |  |
| 12 | 14,6 | 11,8 | 10,2 | 6,0 | 2,4 | 1,4 | 0,7 | 1,4 | 1,8 | 3,5 | 6,8 | 13,4 | 12,5 | 11,2 | 11,3 |  |  |  |  |
| 13 | 32,2 | 31,5 | 34,3 | 33,7 | 34,1 | 32,5 | 33,4 | 34,3 | 33,5 | 34,2 | 34,6 | 35,0 |  |  |  |  |  |  |  |
| 14 | 16,3 | 18,4 | 23,9 | 19,5 | 21,4 | 17,4 | 17,1 | 18,9 | 17,7 | 17,6 | 16,0 | 18,0 | 19,9 | 19,5 | 20,4 | 19,8 |  |  |  |
| 15 | 82,3 | 49,0 | 48,8 | 45,5 | 50,6 | 39,5 | 46,2 | 42,9 | 32,2 | 32,9 | 50,7 | 30,6 | 63,6 | 52,1 | 57,2 | 59,0 |  |  |  |
| 16 | 0,5 | 0,6 | 0,8 | 1,0 | 1,8 | 1,9 | 2,0 | 1,7 | 2,0 | 1,6 | 1,0 | 0,7 | 0,7 | 0,7 | 0,9 | 1,2 | 1,5 | 1,6 | 1,8 |
| 17 | 4,3 | 4,5 | 4,7 | 4,9 | 4,8 | 4,7 | 4,7 | 2,6 | 4,1 | 4,3 | 3,6 | 4,8 | 4,4 | 5,0 | 3,7 | 5,5 |  |  |  |
| 18 | 26,7 | 23,6 | 24,9 | 22,9 | 24,0 | 30,9 | 33,8 | 36,7 | 39,1 | 36,5 | 47,3 | 59,8 | 64,8 | 77,4 |  |  |  |  |  |
| 19 | 136,3 | 124,4 | 137,7 | 133,4 | 137,0 | 130,0 | 135,3 | 136,4 | 134,4 | 139,3 | 134,5 | 141,0 |  |  |  |  |  |  |  |
| 20 | 95,7 | 99,4 | 96,8 | 93,8 | 93,2 | 91,5 | 91,2 | 92,3 | 91,5 | 95,8 | 99,3 | 96,8 | 100,4 | 101,6 |  |  |  |  |  |
| 21 | 26,0 | 24,0 | 28,0 | 17,0 | 22,0 | 15,0 | 26,0 | 27,0 | 27,5 | 28,5 | 30,0 | 27,0 | 27,0 | 28,0 | 30,0 | 31,1 | 35,0 | 34,8 |  |
| 22 | 8,3 | 7,4 | 14,6 | 10,9 | 8,5 | 9,8 | 13,0 | 12,6 | 11,4 | 13,8 | 10,6 | 8,0 | 7,2 | 6,4 | 12,5 | 14,3 | 13,4 | 12,8 | 15,0 |
| 23 | 62,1 | 151,9 | 165,9 | 134,2 | 82,7 | 110,7 | 74,1 | 51,1 | 118,3 | 89,1 | 46,2 | 93,7 | 80,7 | 51,7 | 118,1 |  |  |  |  |
| 24 | 8,3 | 7,4 | 14,6 | 10,9 | 8,5 | 9,8 | 13,0 | 12,6 | 11,4 | 13,8 | 10,6 | 8,0 | 7,2 | 6,4 |  |  |  |  |  |
| 25 | 221 | 126 | 373 | 284 | 287 | 263 | 226 | 280 | 223 | 250 | 183 | 220 | 231 | 321 | 309 | 299 | 236 | 218 |  |
| 26 | 35,1 | 33,3 | 35,4 | 33,0 | 32,7 | 40,5 | 41,7 | 51,6 | 56,7 | 106,9 | 107,9 |  |  |  |  |  |  |  |  |
| 27 | 253 | 287 | 453 | 728 | 599 | 557 | 759 | 718 | 573 | 587 | 472 | 427 | 493 | 601 | 547 | 809 |  |  |  |
| 28 | 829 | 523 | 775 | 615 | 448 | 477 | 535 | 487 | 513 | 334 | 296 | 937 | 486 | 543 | 543 | 467 | 456 | 459 |  |
| 29 | 701 | 709 | 780 | 861 | 786 | 764 | 886 | 844 | 318 | 806 | 673 | 788 | 583 | 560 | 674 | 660 | 602 |  |  |
| 30 | 1297 | 1207 | 1548 | 1523 | 1321 | 1114 | 1306 | 1170 | 1532 | 1338 | 1282 | 1641 | 1565 | 1771 | 1940 |  |  |  |  |
| 31 | 12 328 | 11 624 | 14 196 | 10 974 | 8973 | 10 841 | 7528 | 17 651 | 20 618 |  |  |  |  |  |  |  |  |  |  |
| 32 | 1530 | 1535 | 1435 | 1284 | 1125 | 1258 | 1364 | 2080 | 2053 | 2581 | 2574 | 3203 |  |  |  |  |  |  |  |

**Практическая работа №35 «Решение матричной игры методом итераций»**

**Цель**. Научиться выбирать оптимальную стратегию игры

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

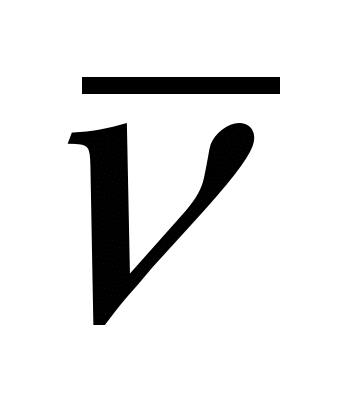
**Метод итераций**– один из самых простых численных методов решения игр (приближённый метод). Идея метода: разыгрывается «мысленный эксперимент», в котором А и В поочерёдно применяют друг против друга свои стратегии, стремясь выиграть побольше (проиграть поменьше). Эксперимент состоит из ряда «партий» игры. Игрок А выбирает произвольно одну из своих стратегий *Ai*. Противник отвечает ему той из своих стратегий *Bj*, которая хуже всего для А при стратегии *Ai*. Далее А выбирает ту стратегию *Ak*, которая даёт ему максимальный выигрыш при стратегии *Bj*. Теперь противник отвечает той стратегией, которая является наихудшей для нашей смешанной стратегии *(Ai , Ak)* , в которой до сих пор применённые стратегии встречаются с равными вероятностями=1/2. И так далее: на каждом шаге итерационного процесса каждый игрок отвечает на очередной ход другого той стратегией, которая является оптимальной для него относительно смешанной стратегии другого, в которую все применённые ранее стратегии входят пропорционально частотам их применения. Такой метод сходится: при увеличении числа партий средний выигрыш на 1 партию будет стремиться к цене игры, а частоты применения стратегий – к вероятностям в оптимальных смешанных стратегиях.  
  
ПРИМЕР. Рассмотрим матричную игру.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | В1 | В2 | В3 | min |
| A1 | *7* | *2* | *9* | *2* |
| A2 | *2* | *9* | *0* | *0* |
| А3 | *9* | *0* | *11* | *0* |
| max | *9* | *9* | *11* |  |

Проведем мысленный эксперимент:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *k* | *i* | *В1* | *В2* | *В3* | *J* | *A1* | *A2* | *А3* | *v* |  | *v\** |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | 3  2  2  2  1  3  1  2  2  1  3  2  2  1  3 | *9*  *11*  *13*  *15*  *22*  *31*  *и* | *0*  *9*  *18*  *27*  *29*  *29*  *т.д.* | *11*  *11*  *11*  *11*  *20*  *31* | 2  2  3  3  3  2  2  2  3  1  2  2  3  1  2 | 2  4  13  **22**  31  **33** | **9**  **18**  **18**  18  18  27 | 0  0  11  **22**  **33**  **33** | 0  4.5  3.67  2.75  4.0  4.84 | 9   6  5.5  6.6  5.5 | 4.5  6.75  4.84  4.13  5.3  5.17  4.79  5.3  4.78  5.1  4.87  5.2  4.84  5.07  4.9 |

*v* – нижняя оценка игры, равная минимальному накопленному выигрышу, делённому на число партий. Аналогично – верхняя оценка*. v\* –*среднее арифметическое между оценками.  
  
*v\** ≈5, p1\*=4/15≈0.266, p2\*=7/15≈0.468, p3\*=4/15≈0.266  
  
q1\*=2/15≈0.133, q2\*=8/15≈0.534, q3\*=5/15≈0.333  
  
**Задание**. 



1. Найти нижнюю и верхнюю цену игры по принципу максимина (минимакса)
2. Провести итерационный процесс игры из 20 партий.
3. Найти оптимальные смешанные стратегии для каждого игрока и оценить цену игры

**Последовательность выполнения работы**  
  
Запустить Excel

* 1. Сформировать матрицу игры 4х3 с целочисленными выигрышами, используя генератор случайных чисел **СЛЧИС()**

PS. Диапазон случайных чисел [0,10]. Случайное число округлить до целых. Заменить формулу на случайное число с помощью клавиши F9.

* 1. Найти нижнюю и верхнюю цену игры по принципу максимина (минимакса)
  2. Подготовить таблицу для метода итераций
  3. Провести итерационный процесс игры из 20 партий
  4. Подсчитать количество применений игроками каждой стратегии
  5. Вычислить вероятность (относительную долю) применения игроками каждой стратегии
  6. Найти приближенное значение цены игры, как среднего арифметического выигрышей в последних пяти партиях

**Практическое занятие № 36**  
**Тема**: Моделирование систем массового обслуживания

**Цель**: Научиться оценивать СМО по статистическим данным и находить пути улучшения СМО

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

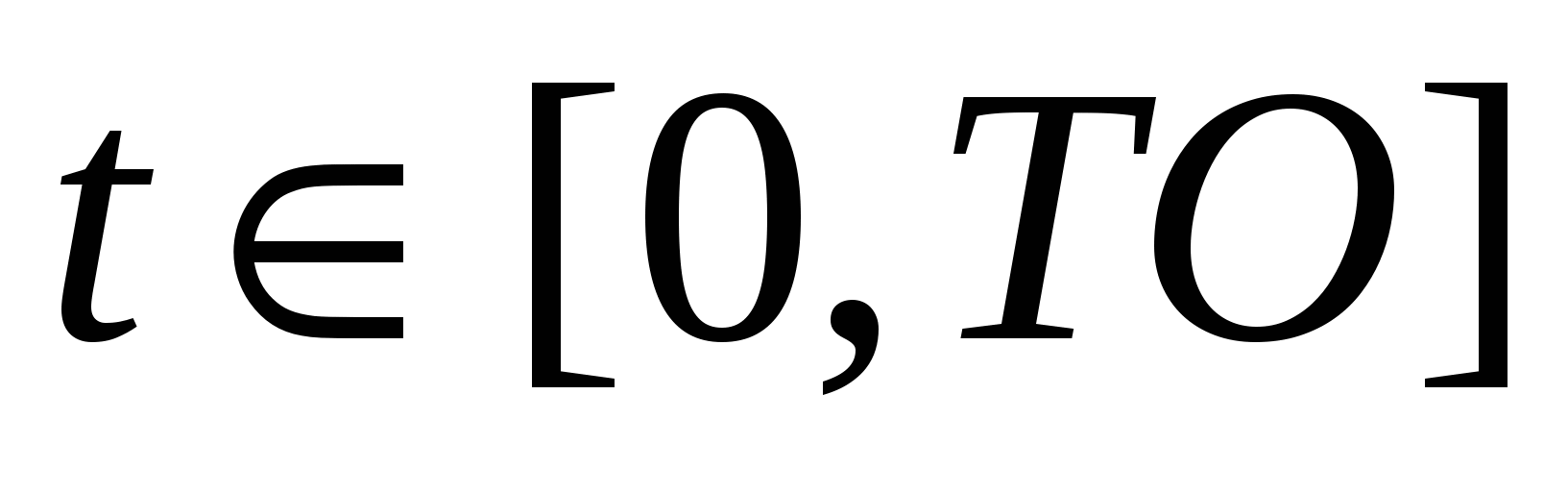
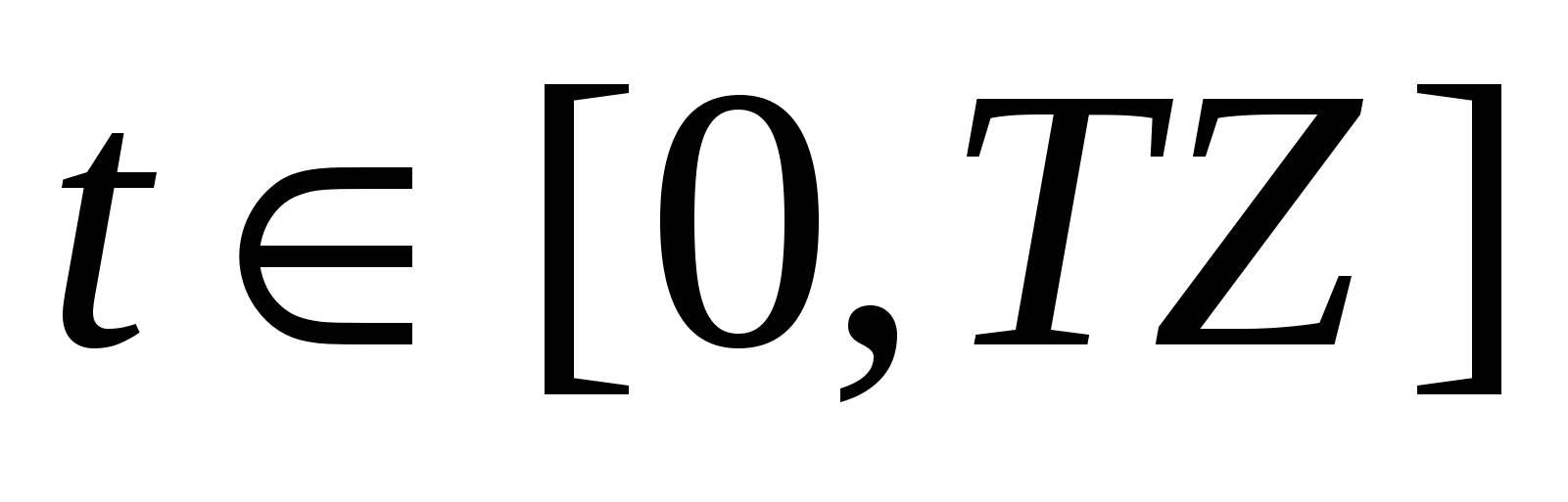
−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

**Задание**. 

1. Смоделировать работу СМО
2. Найти интенсивность поступления и интенсивность обслуживания заявок
3. Проанализировать работу СМО

**Последовательность выполнения работы**  
  
Запустить Excel  
  
1.1. Используя генератор случайных чисел, получить статистические данные:  
  
а) время между заявками   
  
б) время обслуживания   
  
2.1. Вычислить среднее время поступления заявок tз и интенсивность поступления λ=1/tз   
  
2.2. Вычислить среднее время обслуживания tоб и интенсивность обслуживания μ=1/tоб  
  
3.1. Определить тип СМО  
  
3.2. Вычислить основные характеристики СМО   
  
Для СМО с отказами: ρ, P0, Pотк, Q, A, k  
  
Для СМО с очередью: ρ, P0, Lоч, Wоч, Lсист, Wсист  
  
3.3. Вычислить среднюю прибыль P  
  
Для СМО с очередью P= (λ ∙ 3600 -∙ Lоч ) ∙ SZ – SO ∙ n  
  
Для СМО с отказами P=A ∙ SZ ∙ 3600– SO ∙ n  
  
3.4. Предложить пути улучшения работы СМО: 1)сокращение или увеличение количества каналов, 2) уменьшение времени обслуживания  
  
3.5. Проверить, как изменится работоспособность СМО в каждом случае.  
  
**Варианты задач**



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № п/п | Максимальное время между заявками (сек)   ТZ | Максимальное время обслуживания (сек)   ТО | Количество каналов в СМО   n | Средняя прибыль от обслуживания заявки (руб)  SZ | Стоимость содержания канала (руб/час)  SО | Наличие очереди |
|  | 30 | 30 | 1 | 10 | 500 | + |
|  | 30 | 30 | 1 | 10 | 500 | - |
|  | 60 | 60 | 1 | 30 | 500 | + |
|  | 60 | 60 | 1 | 30 | 500 | - |
|  | 30 | 40 | 2 | 20 | 500 | + |
|  | 30 | 40 | 2 | 20 | 500 | - |
|  | 30 | 60 | 2 | 30 | 500 | + |
|  | 30 | 60 | 2 | 30 | 500 | - |
|  | 40 | 60 | 3 | 30 | 500 | + |
|  | 40 | 60 | 3 | 30 | 500 | - |
|  | 40 | 80 | 3 | 50 | 700 | + |
|  | 40 | 80 | 3 | 50 | 700 | - |
|  | 30 | 40 | 1 | 20 | 700 | + |
|  | 30 | 40 | 1 | 20 | 700 | - |
|  | 60 | 70 | 1 | 40 | 1000 | + |
|  | 60 | 70 | 1 | 40 | 1000 | - |
|  | 40 | 40 | 2 | 20 | 1000 | + |
|  | 40 | 40 | 2 | 20 | 1000 | - |
|  | 30 | 70 | 2 | 40 | 700 | + |
|  | 30 | 70 | 2 | 40 | 700 | - |
|  | 50 | 120 | 3 | 100 | 700 | + |
|  | 50 | 120 | 3 | 100 | 700 | - |
|  | 40 | 100 | 3 | 80 | 1000 | + |
|  | 40 | 100 | 3 | 80 | 1000 | - |
|  | 60 | 120 | 3 | 100 | 1000 | + |
|  | 60 | 120 | 3 | 100 | 1000 | - |

**Практическое занятие №37 «Моделирование прогноза»**

**Цель работы**: Построение прогноза временного ряда несколькими способами и выбор лучшей модели прогнозирования.

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

Нужно сделать

• Взять 2 временных ряда -- один в виде отдельного файла, другой из документа Excel

• Провести корреляционный анализ каждого временного ряда – построить его график, рассчитать АКФ, построить график АКФ, определить свойства ряда

• Построить прогнозы заданных временных рядов несколькими различными способами. Модели временных рядов выбирать из текста лекции.

• Оценить ошибки прогнозирования и на основании рассчитанных ошибок выбрать наилучшую модель прогнозирования.

Прогнозирование – одна из самых востребованных задач бизнес-аналитики. Продажи, поставки, заказы – это процессы, распределенные во времени, следовательно, прогнозирование в области продаж, сбыта и спроса, управления материальными запасами и потоками обычно связано именно с анализом временных рядов. Временной ряд – последовательность наблюдений за изменениями во времени значений параметров некоторого объекта или процесса. Временные отсчеты – значения, зафиксированные в некоторые, обычно равноотстоящие моменты времени. В задачах анализа временных рядов мы имеем дело с дискретным временем, когда каждое наблюдение за параметром образует временной отсчет. Все временные отсчеты нумеруются в порядке возрастания. Тогда временной ряд будет представлен в виде X={x1,x2,…,xn}. Одномерные временные ряды содержат наблюдения за изменением только одного параметра исследуемого процесса или объекта, а многомерные – за двумя параметрами или более. Например, трехмерный временной ряд, содержащий наблюдения за тремя параметрами X,Y,Z процесса F можно записать в следующем виде F={(x1,y1,z1), (x2,y2,z2),…,(xn,yn,zn)} Цели и задачи анализа временных рядов • Описание характеристик и закономерностей ряда • Моделирование – построение модели исследуемого процесса • Прогнозирование – предсказание будущих значений временного ряда

Пример. Пусть дан ряд, который содержит последовательность ежемесячных наблюдений за продажами.

Месяц Продажи

Январь 125

Февраль 130

Март 140

Апрель 132

Май 145

Июнь 150

Июль 148

Август 155

Сентябрь 157

Октябрь 160

Ноябрь 158

Декабрь 165

**Практическое занятие №38 «Выбор оптимального решения с помощью дерева решений»**

**Цель: принятие решений с применением дерева решений.**

**Форма отчета:**

−выполнить задание;

−показать преподавателю;

−ответить на вопросы преподавателя.

**Время выполнения: 2 ч**

В постановочном плане рассмотрим несколько задач, которые могут быть решены с помощью данного метода.

Процесс принятия решений с помощью дерева решений в общем случае предполагает выполнение следующих пяти этапов.

Этап 1. Формулирование задачи. Прежде всего необходимо отбросить не относящиеся к проблеме факторы, а среди множества оставшихся выделить существенные и несущественные. Это позволит привести описание задачи принятия решения к поддающейся анализу форме. Должны быть выполнены следующие основные процедуры: определение возможностей сбора информации для экспериментирования и реальных действий; составление перечня событий, которые с определенной вероятностью мо1уг произойти; установление временного порядка расположения событий, в исходах которых содержится полезная и доступная информация, и тех последовательных действий, которые можно предпринять.

Этап 2. *Построение дерева решений.*

Этап 3. Оценка вероятностей состояний среды, т.е. сопоставление шансов возникновения каждого конкретного события. Следует отметить, что указанные вероятности определяются либо на основании имеющейся статистики, либо экспертным путем.

Этап 4. Установление выигрышей (или проигрышей как выигрышей со знаком минус) для каждой возможной комбинации альтернатив (действий) и состояний среды.

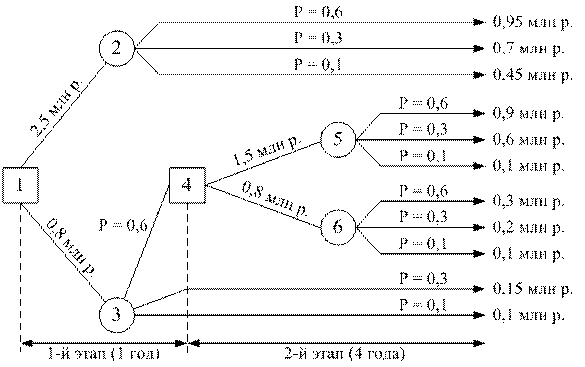
Этап 5. *Решение задачи.*

В зависимости от отношения к риску решение задачи может выполняться с позиций так называемых «объективистов» и «субъективистов». Поясним эти понятия на следующем примере. Пусть предлагается лотерея: за 10 дол. (стоимость лотерейного билета) игрок с равной вероятностью р – 0,5 может ничего не выиграть или выиграть 100 дол. Один индивид пожалеет и 10 дол. за право участия в такой лотерее, т.е. просто не купит лотерейный билет, другой готов заплатить за лотерейный билет 50 дол., а третий заплатит даже 60 дол. за возможность получить 100 дол. (например, когда ситуация складывается так, что, только имея 100 дол., игрок может достичь своей цели, поэтому возможная потеря последних денежных средств, а у него их ровно 60 дол., не меняет для него ситуации).

Безусловным денежным эквивалентом(БДЭ) игры называем максимальная сумма денег, которую ЛПР готов заплатить за участие в игре (лотерее), или, что то же, та минимальная сумма денег, за которую он готов отказаться от игры. Каждый индивид имеет свой БДЭ.

Индивида, для которого БДЭ совпадает с ожидаемой денежной оценкой (ОДО) игры, т.е. со средним выигрышем в игре (лотерее), условно называют объективистом, индивида, для которого БДЭ ≠ ОДО, – субъективистом. Ожидаемая денежная оценка рассчитывается как сумма произведений размеров выигрышей на вероятности этих выигрышей. Например, для нашей лотереи ОДО = 0,5 0 + 0,5 ×100 = 50 дол. Если субъективист склонен к риску, то его БДЭ > ОДО. Если не склонен, то БДЭ < ОДО[47].

Требуется принять решение о замене старого оборудования на новое того же вида или его ремонте. Отремонтированное оборудование впоследствии можно частично заменить на новое, более современное, или отремонтировать его заново.   
Решение определяется будущим спросом на продукцию, которую производят на этом оборудовании.   
Полная замена оборудования экономически оправдана при высоком уровне спроса. С другой стороны, можно отремонтировать старое оборудование и через один год, например, заменить его на новое, более совершенное, или заново его отремонтировать.   
В данной задаче процесс принятия решения состоит из двух этапов: решение в настоящий момент времени о замене или ремонте оборудования и решение, принимаемое через один год, относительно частичной его замены и ремонта.   
Предполагается, что спрос может оказаться высоким, средним и низким.   
Дерево решений имеет два типа вершин: "решающие" и "случайные" (рис.).   
   
Рис. Дерево решений   
Начиная с "решающей" вершины 1, необходимо принять решение о полной замене оборудования или его ремонте.   
Вершины 2 и 3 являются "случайными". Фирма будет рассматривать возможность установления более совершенного оборудования или повторного ремонта старого в том случае, если спрос по истечении одного года установится на высоком уровне. Поэтому в вершине 4 принимается решение о частичной замене старого оборудования более совершенным или ремонте старого. Вершины 5 и 6 "случайные".



Допускается, что фирма рассматривает эту задачу на пятилетний период. Анализ рыночной ситуации показывает, что вероятности высокого, среднего и низкого уровней спроса составляют соответственно 0,6, 0,3 и 0,1. Замена новым оборудованием того же вида, что и старое, обойдется в 2,5 млн р., а ремонт старого – в 0,8 млн р.   
Затраты на частичную замену оборудования более совершенным оцениваются в 1,5 млн р., а повторный ремонт старого – в 0,8 млн р.   
Ежегодные доходы для каждой стратегии фирмы следующие.   
1. Замена старого оборудования на новое того же вида при высоком, среднем и низком уровнях спроса даёт соответственно 0,95, 0,7 и 0,45 млн р.   
2. Ремонт старого оборудования при высоком, среднем и низком уровнях спроса оценивается соответственно в 0,3, 0,15 и 0,1 млн р.   
3. Частичная замена оборудования на более совершенное при высоком, среднем и низком уровнях спроса составит соответственно 0,9, 0,6 и 0,4 млн р.

4. Повторный ремонт старого оборудования при высоком, среднем и низком уровнях спроса предполагает 0,3, 0,2 и 0,1 млн р. соответственно.   
Определим оптимальную стратегию фирмы в замене оборудования.

**Решение**.   
Оценим результаты каждой стратегии и определим, какие решения следует принимать в "решающих" вершинах 1 и 4.   
Вычисления начнем с этапа 2. Для последних 4 лет альтернативы, относящиеся к вершине 4, оцениваются так:   
*ДЧЗ*= (0,9 × 0,6 + 0,6 × 0,3 + 0,4 × 0,1) × 4 – 1,5 = 1,54 млн р.,

ДДР*=*(0,3 × 0,6 + 0,2 × 0,3 + 0,1 × 0,1) × 4 – 0,8 = 0,2 млн р.,   
где ДЧЗ– доход от частичной замены оборудования на более совершенное; ДДР– доход от замены оборудования, прошедшего дважды ремонт.   
Так как ДЧЗ>ДДР, то в вершине 4 выгоднее частично заменить оборудование на более совершенное, при этом доход составит 1,54 млн р.   
Для дальнейших расчетов в вершине 4 можно оставить одну ветвь, которой соответствует доход в 1,54 млн р. за 4 года.   
Вычислим доходы на 1-м этапе для "решающей" вершины 1:   
ДЗН = (0,95 × 0,6 + 0,7 × 0,3 + 0,45 × 0,1) × 5 – 2,5 = 1,625 млн р.,   
ДЗО = 0,3 × 0,6 × 1 + 0,15 × 0,3 × 5 + 0,1 × 0,1 × 5 +1,54 – 0,8 = 1,195 млн р.,   
где ДЗН– доход от замены старого оборудования на новое того же вида; ДЗО— доход от отремонтированного оборудования и дальнейшей замены на более совершенное.   
Так как ДЗН>ДЗО, то оптимальным решением в вершине 1 является полная замена старого оборудования на новое того же вида.   
Итак, оптимальной стратегией фирмы в замене оборудования является полная замена старого оборудования на новое того же вида, при этом доход составит 1,625 млн р.

**Практическая часть**

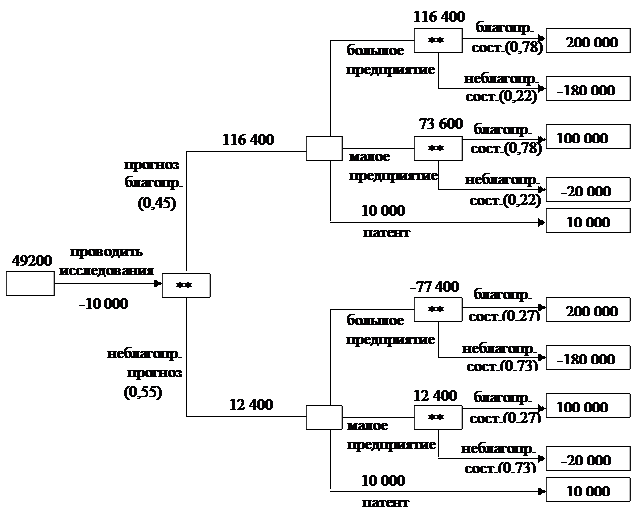
* 1. Фирма планирует построить среднее или малое предприятие по производству пользующейся спросом продукции. Решение о строительстве определяется будущим спросом на продукцию, которую предполагается выпускать на планируемом предприятии.   
     Строительство среднего предприятия экономически оправдано при высоком спросе, но можно построить малое предприятие и через 2 года его расширить.   
     Фирма рассматривает данную задачу на десятилетний период. Анализ рыночной ситуации, проведенный службой маркетинга, показывает, что вероятности высокого  и низкого уровней спроса составляют А=0,65 и В=0,35 соответственно.   
     Строительство среднего предприятия составит С=7,5 млн. руб., малого – D=1,8 млн. руб. Затраты на расширение малого предприятия оценивается в Е=3,4 млн. руб.   
     Ожидаемые ежегодные доходы для каждой из возможных альтернатив:
* среднее предприятие при высоком (низком) спросе – F=1,4 (K=0,38) млн. руб.;
* малое предприятие при низком спросе – L=0,25 млн. руб.;
* малое предприятие при высоком спросе – М=0,27 млн. руб.;
* расширенное предприятие при высоком (низком) спросе дает N=1,6 (P=0,24) млн. руб.;
* малое предприятие без расширения при высоком спросе в течение первых двух лет и последующем низком спросе дает R=0,2 млн. руб. за остальные восемь лет.

Определить оптимальную стратегию фирмы в строительстве предприятий по выпуску продукции.

1. В условиях № 4.9определите, стоит ли компании заказать исследование возможных состояний рынка некоторой фирме, которая оценивает стоимость своих услуг в 10000 руб. Про эту фирму известно, что она утверждает, что ситуация будет благоприятной с вероятностью 0,45 и неблагоприятной с вероятностью 0,55. Прогнозы фирмы оправдываются следующим образом (известно из ранее составленных его прогнозов):

Таблица 4.3

|  |  |
| --- | --- |
| Прогноз фирмы о состоянии рынка | Фактически |
| Благоприятное | Неблагоприятное |
| Благоприятный | 0,78 | 0,22 |
| Неблагоприятный | 0,27 | 0,73 |



**Список литературы**

1. *Лаврищева, Е. М.*Программная инженерия и технологии программирования сложных систем : учебник для вузов / Е. М. Лаврищева. — 2-е изд., испр. и доп. — Москва : Издательство Юрайт, 2019. — 432 с. — (Бакалавр. Академический курс). — ISBN 978-5-534-07604-2. — Текст : электронный // ЭБС Юрайт [сайт]. — URL: https://biblio-online.ru/bcode/436514
2. *Казарин, О. В.*Основы информационной безопасности: надежность и безопасность программного обеспечения : учебное пособие для среднего профессионального образования / О. В. Казарин, И. Б. Шубинский. — Москва : Издательство Юрайт, 2019. — 342 с. — (Профессиональное образование). — ISBN 978-5-534-10671-8. — Текст : электронный // ЭБС Юрайт [сайт]. — URL: https://biblio-online.ru/bcode/431080.
3. *Черткова, Е. А.*Программная инженерия. Визуальное моделирование программных систем : учебник для среднего профессионального образования / Е. А. Черткова. — 2-е изд., испр. и доп. — Москва : Издательство Юрайт, 2019. — 147 с. — (Профессиональное образование). — ISBN 978-5-534-09823-5. — Текст : электронный // ЭБС Юрайт [сайт]. — URL: https://biblio-online.ru/bcode/441255.
4. *Черткова, Е. А.*Программная инженерия. Визуальное моделирование программных систем : учебник для среднего профессионального образования / Е. А. Черткова. — 2-е изд., испр. и доп. — Москва : Издательство Юрайт, 2019. — 147 с. — (Профессиональное образование). — ISBN 978-5-534-09823-5. — Текст : электронный // ЭБС Юрайт [сайт]. — URL: https://biblio-online.ru/bcode/441255
5. *Куприянов, Д. В.*Информационное обеспечение профессиональной деятельности : учебник и практикум для среднего профессионального образования / Д. В. Куприянов. — Москва : Издательство Юрайт, 2019. — 255 с. — (Профессиональное образование). — ISBN 978-5-534-00973-6. — Текст : электронный // ЭБС Юрайт [сайт]. — URL: https://biblio-online.ru/bcode/434578
6. *Казарин, О. В.*Программно-аппаратные средства защиты информации. Защита программного обеспечения : учебник и практикум для вузов / О. В. Казарин, А. С. Забабурин. — Москва : Издательство Юрайт, 2019. — 312 с. — (Специалист). — ISBN 978-5-9916-9043-0. — Текст : электронный // ЭБС Юрайт [сайт]. — URL: https://biblio-online.ru/bcode/437163