Оглавление

1. Организация ЭВМ и организация вычислений: архитектура фон Неймана, влияние машинной архитектуры на языки программирования. 3

2. Императивные языки программирования: характеристики стиля программирования и обзор распространенных императивных языков. 4

3. Функциональная парадигма в программировании. Внутренний параллелизм. 5

4. Объектно-ориентированная парадигма в программировании: базовые принципы, обзор объектно-ориентированных языков программирования. 6

5. Типизация в языках программирования. Механизмы абстракции. 7

6. Синтаксис и семантика языков программирования 9

7. Теория автоматов: основные понятия, конечные автоматы 10

8. Компиляция и интерпретация. Интерпретация непосредственно и с промежуточной компиляцией. Время связывания. Библиотека времени исполнения. 11

9. Компиляторы: архитектура, фазы компиляции, методы синтаксического анализа. 13

10. Фундаментальные структуры данных в программировании 15

11. Графы и деревья: основные понятия, алгоритмы нахождения кратчайших путей. 16

12. Реляционная теория. Отношения. Реляционная алгебра. Теория нормализации. Реляционное исчисление. Целостность. Определение реляционной модели. 17

13. Реляционные базы данных. Проектирование реляционных баз данных. Инфологическое проектирование. Построение ER-моделей и их преобразование в структуру баз данных. Проектирование физической структуры базы данных. Индексирование. 18

14. Теория распределенных систем К.Дейта. Распределенные базы данных. 19

15. Базы знаний. Принципы разработки интеллектуальных систем. 20

16. Языки логического программирования: исчисление предикатов, метод резолюций, декларативная семантика. 22

17. Низкоуровневое программирование: система команд ЭВМ: регистры процессора, режимы адресации. Ассемблер. 23

18. Операционные системы: архитектура, структура классического ядра ОС. 25

19. Функции ОС по управлению памятью, механизмы выделения памяти процессу, управление адресным пространством процесса. 26

20. Распределение процессорного времени в многозадачных системах. Потоки. Нити. Разделение ресурсов при мультипрограммировании. 28

21. Одноранговые сети и сети на основе серверов, домены. Протоколы и стандарты технологий локальных сетей. 29

22. Маршрутизация в сетях ЭВМ. Межсетевое взаимодействие на базе TCP/IP. 30

23. Глобальные сети. Глобальные сети на выделенных линиях. Глобальные сети с коммутацией каналов. Глобальные сети с коммутацией пакетов. 32

24. Сетевые операционные системы, функциональные компоненты. Сетевые службы и сервисы. Типичные сетевые службы и используемые протоколы. 33

25. Сетевые файловые системы, файловые серверы с сохранением состояния и без, кэширование, репликация. Служба каталогов, Межсетевое взаимодействие. 34

26. Высокопроизводительные коммуникационные среды. Обзор распространенных коммуникационных сред. 36

27. Виды параллелизма. Пиковая и реальные производительности. Эффективность. Законы Амдаля. Классификация многопроцессрных ЭВМ. 37

28. Многопроцессорные ЭВМ и многомашинные комплексы. Обзор разных типов доступа к памяти. Общая память, распределенная память, MPP, кластеры. 38

29. Параллельная реализация прямых и итерационных методов решения линейных систем: метод Гаусса, LU-разложение для симметричных матриц. 39

30. Технологии параллельного программирования. Программные средства реализации параллельных вычислений на многопроцессорных ЭВМ. 41

31. Эквивалентные преобразования программ в параллельном программировании. 43

32. Основные дифференциальные уравнения матфизики: уравнение колебаний, уравнение диффузии, уравнение переноса, уравнения газо- и гидродинамики, уравнение Максвелла, уравнение Шредингера, уравнение Клейна-Гордона-Фока и уравнение Дирака. 44

33. Общая, каноническая и основная задача линейного программирования. Графическое решение задачи. Симплекс-метод и симплекс-таблицы. Выбор начального допустимого решения. Двойственность в линейном программировании. Анализ на чувствительность. 46

34. Общая задача нелинейного программирования. Выпуклые множества и выпуклые функции. Задача выпуклого программирования. Условия оптимальности в задачах выпуклого программирования. Методы условной оптимизации. Методы штрафных и барьерных функций. Методы проекции градиента, приведенного градиента, условного градиента. Метод проекции точки на множество. 49

35. Интерполяция и приближение. Постановка задачи. Интерполяционный многочлен Лагранжа. Разделенные разности и интерполяционная формула Ньютона. Интерполяционные сплайны. Погрешность интерполяционных формул....51

36. Методология автоматизированного проектирования. Блочно-иерархический подход к проектированию: декомпозиция, блочность, итерационность, унификация и стандартизация. Иерархические уровни проектирования. Восходящее, нисходящее, смешанное проектирование....53

Уровни проектирования 54

37. Классическое исчисление высказываний. Аксиомы и правила вывода. Вывод формул и вывод формул из гипотез. Теоремы о дедукции. Теоремы полноты и непротиворечивости....55

38. Предикаты и кванторы. Предикатные формулы. Аксиомы и правила вывода. Вывод предикатных формул и вывод предикатных формул из гипотез....57

39. Определение метрического пространства и их примеры. Открытые и замкнутые множества. Полные метрические пространства, их свойства. Пополнение метрических пространств. 59

40. Линейные пространства. Пространства операторов и функционалов. Нормированные и полные нормированные (банаховы) пространства. Норма линейного оператора и функционала в нормированном пространстве....60

41. Одномерные случайные величины. Функция распределения и ее свойства. Дискретные случайные величины и их свойства. Биномиальное распределение. Распределение Пуассона. Равномерное распределение. Нормальное распределение....61

43. Числовые характеристики случайных величин: математическое ожидание и его свойства; дисперсия и ее свойства; ковариация и ее свойства.; коэффициент корреляции и его свойства. Связь между коррелированными и зависимыми случайными величинами. Ковариационная и корреляционная матрицы. Числовые характеристики основных законов распределения. 63

44. Обработка экспериментальных данных. Сглаживание экспериментальных зависимостей. Метод наименьших квадратов. Линейная регрессионная модель. Статистический анализ регрессионной модели...65

45....66

50. Матрицы. Специальные виды матриц. Линейные операции над матрицами и их свойства. Транспонирование матрицы. Умножение матриц и его свойства. Прямое сложение матриц и его свойства. Определитель матрицы. Обратная матрица. Ранг матрицы. Решение матричного уравнения AX=B, где A – невырожденная матрица....67

1. Организация ЭВМ и организация вычислений: архитектура фон Неймана, влияние машинной архитектуры на языки программирования.

Архитектура фон Неймана

Классическая архитектура построения компьютера, в которой выделены: оперативная, последовательно адресуемая память, где хранятся как данные, так и сама программа; процессор, последовательно выполняющий команды из программы. Большинство компьютеров в настоящее время имеют эту архитектуру. Примером другой архитектуры могут служить многопроцессорные компьютеры с параллельными вычислениями. Название дано в честь одного из разработчиков данной архитектуры, известного математика Джона фон Неймана.

«Принципы фон Неймана».

1. Принцип использования двоичной системы счисления для представления данных и команд.
2. Принцип однородности памяти.
   Как программы (команды), так и данные хранятся в одной и той же памяти (и кодируются в одной и той же системе счисления — чаще всего двоичной). Над командами можно выполнять такие же действия, как и над данными.
3. Принцип адресуемости памяти.
   Структурно основная память состоит из пронумерованных ячеек; процессору в произвольный момент времени доступна любая ячейка.
4. Принцип последовательного программного управления
   Все команды располагаются в памяти и выполняются последовательно, одна после завершения другой.
5. Принцип условного перехода.

Влияние машинной архитектуры на языки программирования

Ранние языки программирования разрабатывались так, чтобы написанные на них программы могли эффективно работать на дорогостоящих вычислительных машинах. Поэтому программы, написанные на ранних языках, транслировались в эффективный машинный код, даже если писать их было достаточно сложно. Теперь ситуация изменилась — машины стали работать гораздо быстрее, их стоимость снизилась, но зато возросла стоимость работы программиста. В настоящее время приоритетное значение имеет создание программ, при написании которых меньше вероятность допустить ошибку, даже если при этом несколько возрастет время их выполнения. Например, определение пользовательских типов данных в ML, объект класса (class) в C++ и спецификация пакета (package) в языке Ada упрощают процесс написания корректно работающих программ, но за это приходится расплачиваться уменьшением скорости выполнения.

При разработке языка программирования учитывается архитектура программного обеспечения, которая, соответственно, оказывает влияние на идеологию создаваемого языка. Это влияние определяется:

1) реальным компьютером, на котором будут выполняться написанные на данном языке программы;

2) моделью выполнения, или виртуальным компьютером, который поддерживает этот язык на реальном компьютере.

Среди конструкций языка в этом смысле можно выделить классы:

1. Машинно-зависимые, или элементы, определяемые машиной. При создании таких языков имеется в виду их реализация на какой-то конкретной машине. Примером может служить FORTRAN, в котором есть некоторые крайне неудобные для программиста, но “удобные” для машины конструкции.

2. Элементы, отображаемые на машину. Некоторые языковые конструкции рассчитаны на совершенно определенные способы реализации их в “железе”. Это не значит, что их нельзя реализовать как-то иначе, но авторы языков часто имеют в виду конкретные варианты перевода на машинный язык.

3. Машинно-независимые, или элементы, определяющие виртуальную машину не предусматривают какой-то простой реализации в компьютере. Фактически, такие элементы требуют определенного объема программного моделирования, создания виртуальной машины, которая уже позволяет реализовать эти особенности.

4. Неисполняемые элементы. Они непосредственно не вызывают порождение какого-то машинного кода, но нужны для решения задач, которые сам же язык и ставит перед программистом.

5. Так называемый “syntaxsugar”. Означает возможость записать какое-то часто встречающееся сочетание операторов иным способом, проще и нагляднее.

Syntaxsugar часто встречается в современных языках, поскольку облегчает жизнь и красиво смотрится.

2. Императивные языки программирования: характеристики стиля программирования и обзор распространенных императивных языков.

Императи́вное программи́рование — это парадигма программирования, которая, описывает процесс вычисления в виде последовательности инструкций, изменяющих состояние программы.
Императивные языки программирования противопоставляются функциональным и
логическим языкам программирования. Функциональные языки, например, Haskell, не
представляют собой последовательность инструкций и не имеют глобального состояния. Логические языки программирования, такие как Prolog, обычно определяют что надо вычислить, а не как это надо делать.
Процедурное (императивное) программирование является отражением архитектуры традиционных ЭВМ, которая была предложена фон Нейманом в 1940-х годах. Теоретической моделью процедурного программирования служит алгоритмическая система под названием Машина Тьюринга.

Программа на процедурном языке программирования состоит из последовательности операторов (инструкций), задающих процедуру решения задачи. Основным является оператор присваивания, служащий для изменения содержимого областей памяти.
Концепция памяти как хранилища значений, содержимое которого может обновляться операторами программы, является фундаментальной в императивном программировании.
Выполнение программы сводится к последовательному выполнению операторов с целью преобразования исходного состояния памяти, то есть значений исходных данных, в заключительное, то есть в результаты. Таким образом, с точки зрения программиста имеются программа и память, причем первая последовательно обновляет содержимое последней. Процедурный язык программирования предоставляет возможность программисту определять каждый шаг в процессе решения задачи. Особенность таких языков программирования состоит в том, что задачи разбиваются на шаги и решаются шаг за шагом. Используя процедурный язык, программист определяет языковые конструкции для выполнения последовательности алгоритмических шагов.

Процедурные языки программирования
C, Basic, FORTRAN, Pascal.

3. Функциональная парадигма в программировании. Внутренний параллелизм.

Языки функционального программирования (Functional Languages, Applicative Languages) - класс языков программирования и подкласс декларативных языков, основанных на идеях лямбда-исчисления и теории рекурсивных функций. Они основаны на понятии функции, - описания зависимости результата от аргументов с помощью других функций и элементарных операций. В функциональных языках нет понятия переменной и присваивания, поэтому значение функции зависит только от ее аргументов и не зависит от порядка вычислений. Функциональная программа состоит из совокупности определений функций. Функции, в свою очередь, представляют собой вызовы других функций и предложений, управляющих последовательностью вызовов. Вычисления начинаются с вызова некоторой функции, которая в свою очередь вызывает функции, входящие в ее определение и т.д. в соответствии с иерархией определений и структурой условных предложений. Функции часто либо прямо, либо опосредованно вызывают сами себя. Каждый вызов возвращает некоторое значение в вызвавшую его функцию, вычисление которой после этого продолжается; этот процесс повторяется до тех пор, пока запустившая вычисления функция не вернет конечный результат пользователю. "Чистое" функциональное программирование не признает присваиваний и передач управления. Разветвление вычислений основано на механизме обработки аргументов условного предложения. Повторные вычисления осуществляются посредством рекурсии, являющейся основным средством функционального программирования.
На практике отличие математической функции от понятия «функции» в императивном программировании заключается в том, что императивные функции взаимодействуют и изменяют уже определённые данные. Таким образом, в императивном программировании, при вызове одной и той же функции с одинаковыми параметрами можно получить разные данные на выходе, из-за влияния на функцию внешних факторов. А в функциональном языке при вызове функции с одними и теми же аргументами мы всегда получим одинаковый результат в обоих случаях, входные данные не могут измениться, выходные данные зависят только от них.
• краткость и простота;
Программы на функциональных языках обычно намного короче и проще, чем те же самые
программы на императивных языках.
• строгая типизация;
• функции — объекты вычисления;
В функциональных языках (равно как и вообще в языках программирования и математике) функции могут быть переданы другим функциям в качестве аргумента или возвращены в качестве результата. Функции, принимающие функциональные аргументы, называются функциями высших порядков или функционалами.
• чистота (отсутствие побочных эффектов);
• отложенные (ленивые) вычисления.
В традиционных языках программирования (например, C++) вызов функции приводит к вычислению всех аргументов. Этот метод вызова функции называется вызов-по-значению. Если какой-либо аргумент не использовался в функции, то результат вычислений пропадает, следовательно, вычисления были произведены впустую. В каком-то смысле противоположностью вызова-по-значению является вызов-по-необходимости (ленивые вычисления). В этом случае аргумент вычисляется, только если он нужен для вычисления результата.

Haskell, LISP, ML/SML, Nemerle, F# .
Внутренний параллелизм: подразумевается, что если исходный алгоритм был численно устойчив, то он останется таким же и в параллельной форме. Преимуществом функциональных программ является то, что они предоставляют широчайшие возможности для автоматического распараллеливания вычислений. Поскольку отсутствие побочных эффектов гарантировано, в любом вызове функции всегда допустимо параллельное вычисление двух различных параметров — порядок их вычисления не может оказать влияния на результат вызова.

4. Объектно-ориентированная парадигма в программировании: базовые принципы, обзор объектно-ориентированных языков программирования.

Объектно-ориентированная парадигма в программировании, парадигма, описывающая процесс вычислений как взаимодействие некоторого набора объектов.
Класс — это тип, описывающий устройство объектов. Понятие «класс» подразумевает некоторое поведение и способ представления. Понятие «объект» подразумевает нечто, что обладает определённым поведением и способом представления. Говорят, что объект — это экземпляр класса.
Первым широко распространённым объектно-ориентированным языком программирования стал Smalltalk.
Инкапсуляция — это принцип, согласно которому любой класс должен рассматриваться как чёрный ящик — пользователь класса должен видеть и использовать только интерфейсную часть класса (т. е. список декларируемых свойств и методов класса) и не знать о его внутренней реализации. Поэтому данные принято инкапсулировать в классе таким образом, чтобы доступ к ним по чтению или записи осуществлялся не напрямую, а с помощью методов. Принцип инкапсуляции (теоретически) позволяет минимизировать число связей между классами и, соответственно, упростить независимую реализацию и модификацию классов.
Наследованием называется возможность порождать один класс от другого с сохранением всех свойств и методов класса-предка (прародителя, иногда его называют суперклассом) и добавляя, при необходимости, новые свойства и методы. Набор классов, связанных отношением наследования, называют иерархией. Наследование призвано отобразить такое свойство реального мира, как иерархичность.
Полиморфизмом называют явление, при котором функции (методу) с одним и тем же именем соответствует разный программный код (полиморфный код) в зависимости от того, объект какого класса используется при вызове данного метода. Полиморфизм обеспечивается тем, что в классе-потомке изменяют реализацию метода класса-предка с обязательным сохранением сигнатуры метода. Это обеспечивает сохранение неизменным интерфейса класса-предка и позволяет осуществить связывание имени метода в коде с разными классами — из объекта какого класса осуществляется вызов, из того класса и берётся метод с данным именем. Такой механизм называется динамическим (или поздним) связыванием — в отличие от статического (раннего) связывания, осуществляемого на этапе компиляции.
Принципы:
1. Всё является объектом.
2. Вычисления осуществляются путём взаимодействия (обмена данными) между объектами, при котором один объект требует, чтобы другой объект выполнил некоторое действие. Объекты взаимодействуют, посылая и получая сообщения.
Сообщение — это запрос на выполнение действия, дополненный набором аргументов, которые могут понадобиться при выполнении действия.
3. Каждый объект имеет независимую память, которая состоит из других объектов.
4. Каждый объект является представителем (экземпляром) класса, который выражает общие свойства объектов.
5. В классе задаётся поведение (функциональность) объекта. Тем самым все объекты, которые являются экземплярами одного класса, могут выполнять одни и те же действия.
6. Классы организованы в единую древовидную структуру с общим корнем, называемую иерархией наследования. Память и поведение, связанное с экземплярами определённого класса, автоматически доступны любому классу, расположенному ниже в иерархическом дереве.
C#, C++, Java, Objective C, Ruby, Smalltalk

5. Типизация в языках программирования. Механизмы абстракции.

Типизация - это способ защититься от использования объектов одного класса вместо другого, или по крайней мере управлять таким использованием.

Типы данных различаются начиная с нижних уровней системы. Так, например, даже в Ассемблере x86 различаются типы «целое число» и «вещественное число». Это объясняется тем, что для чисел рассматриваемых типов отводятся различные объёмы памяти, используются различные регистры микропроцессора, а для операций с ними применяются различные команды Ассемблера и различные ядра микропроцессора.

Концепция типа данных появилась в языках программирования высокого уровня как естественное отражение того факта, что обрабатываемые программой данные могут иметь различные множества допустимых значений, храниться в памяти компьютера различным образом, занимать различные объёмы памяти и обрабатываться с помощью различных команд процессора. Преимущества от использования типов данных:

• Надёжность. Типы данных защищают от трёх видов ошибок:

1. Некорректное присваивание.
2. Некорректная операция. Позволяет избежать попыток применения выражений вида «Hello world» + 1.
3. Некорректная передача параметров.

• Стандартизация.

Контроль типов и системы типизации

• Статическая типизация — контроль типов осуществляется при компиляции.

• Динамическая типизация — контроль типов осуществляется во время выполнения.

Контроль типов также может быть строгим и слабым.

• Строгая типизация — совместимость типов автоматически контролируется транслятором:

• Номинативная типизация — совместимость должна быть явно указана (наследована) при определении типа.

• Структурная типизация — совместимость определяется структурой самого типа (типами элементов, из которых построен составной тип).

• Слабая типизация — совместимость типов никак транслятором не контролируется. В языках со слабой типизацией обычно используется подход под названием «утиная типизация» — когда совместимость определяется и реализуется общим интерфейсом доступа к данным типа.

Стати́ческая типиза́ция — приём, широко используемый в языках программирования, при котором переменная, параметр подпрограммы, возвращаемое значение функции связывается с типом в момент объявления и тип не может быть изменён позже. Примеры статически типизированных языков — Ада, Си++, Паскаль.

Динами́ческая типиза́ция — приём, широко используемый в языках программирования и языках спецификации, при котором переменная связывается с типом в момент присваивания значения, а не в момент объявления переменной. Таким образом, в различных участках программы одна и та же переменная может принимать значения разных типов.

+:

• Минимум дополнительных строк: переменные надо либо просто объявить (JavaScript), либо вообще объявлять не обязательно (PHP).

• Пропадают операции приведения типа:

• Соответственно, упрощается написание простых программ.

• Повышается гибкость языка.

• Ускоряет работу компилятора — а значит, производственный цикл «написать-проверить».

-:

• Статическая типизация позволяет уже при компиляции заметить простые ошибки «по недосмотру». Для динамической типизации требуется как минимум выполнить данный участок кода.

• Для написания сложного кода нужна особая культура программирования: венгерская нотация, юнит-тестирование и т. д.

• Интерфейсная часть модуля в статическом языке является существенной частью документации — а изредка вообще позволяет обойтись без документирования.

• Низкая скорость, связанная с динамической проверкой типа. Большинство языков с динамической типизацией интерпретируемые, а не компилируемые.

Строгая типизация — один из вариантов политики работы с типами данных, которая используется в языках программирования. Строгая типизация подразумевает выполнение следующих обязательных условий:

1. Любой объект данных (переменная, константа, выражение) в языке всегда имеет строго определённый тип, который фиксируется на момент компиляции программы (статическая типизация) или определяется во время выполнения (динамическая типизация).
2. Допускается присваивание переменной только значения, имеющего строго тот же тип данных, что и переменная, те же ограничения действуют в отношении передачи параметров и возврата результатов функций.
3. Каждая операция требует параметров строго определённых типов.

4. Неявное преобразование типов не допускается. Если в программе необходимо присвоить значение одного типа данных переменной другого типа, это можно сделать, но только путём явного применения специальной операции преобразования типа, которая в таких случаях обычно является частью языка программирования.
Латентная, неявная или утиная типизация — вид динамической типизации, применяемой в некоторых языках программирования, когда границы использования объекта определяются его текущим набором методов и свойств, в противоположность наследованию от определённого класса. То есть считается, что объект реализует интерфейс, если он содержит все методы этого интерфейса, независимо от связей в иерархии наследования и принадлежности к какому-либо конкретному классу.
Название термина пошло от английского «duck test» («тест на утку»), который в оригинале
звучит как:
«If it walks like a duck and quacks like a duck, it must be a duck»
Утиная типизация решает такие проблемы иерархической типизации, как:
• невозможность явно указать (путем наследования) на совместимость интерфейса со всеми настоящими и будущими интерфейсами, с которыми он идейно совместим
• чрезмерное увеличение числа связей в иерархии типов при хотя бы частичной попытке это сделать.

6. Синтаксис и семантика языков программирования

Синтаксис — сторона языка программирования, которая описывает структуру программ как наборов символов (обычно говорят — безотносительно к содержанию). Синтаксису языка противопоставляется его семантика. Синтаксис языка описывает «чистый» язык, в то же время семантика приписывает значения (действия) различным синтаксическим конструкциям.

Сема́нтика — в программировании — система правил определения поведения отдельных языковых конструкций. Семантика определяет смысловое значение предложений алгоритмического языка

Каждый язык программировния имеет синтаксическое описание. Обычно синтаксис языка определяют посредством правил Бэкуса-Наура.
Форма Бэкуса-Наура — формальная система описания синтаксиса, в которой одни синтаксические категории последовательно определяются через другие категории. Метапеременные представляют собой слова или группы слов, заключенных в угловые скобки. Под значением метапеременной понимается некоторая конечная последовательность основных символов языка, из которых, в конечном счете, состоят программы. Символ : := означает "определяется как", символ | означает "или", символ * - "произвольное количество повторений (в том числе ноль раз) того символа, за которым он указан". Символы, указанные в квадратных скобках, являются необязательными

Способы описания семантики. Обычный способ описания семантики (естественный язык) хорош до тех пор, пока эти описания читают люди. Но встречаются ситуации, когда описание должно быть доступно компьютеру, например, при трансляции программы с данного языка программирования. В этом случае подобные описания неприемлемы. Именно это и послужило стимулом к разработке теории, которая бы позволила осуществить формализацию семантики. Задача формализации семантики языка программирования, однако, оказалась гораздо сложнее, чем задача формализации его синтаксиса.

Кратко опишем наиболее распространённые способы формального описания семантики.
Для описания семантики языка необходимо ввести в рассмотрение правила перехода от одних выражений языка к некоторым другим выражениям, несущим ту же информацию (имеющим тот же смысл), т.е. правила преобразования языковых выражений. Смысл языкового выражения рассматривается здесь как то общее, что имеется у данного выражения и всех других синонимичных ему выражений того же или какого-либо другого (не обязательно естественного) языка.

При таком подходе выражение "описать семантику языка L" означает: (1) указать другой язык L', на который будут переводиться выражения языка L; (2) задать правила перевода любых выражений с языка L на язык L' и обратно. Язык L' называется семантическим языком.

7. Теория автоматов: основные понятия, конечные автоматы

Абстра́ктный автома́т — математическая абстракция, модель дискретного устройства, имеющего один вход, один выход и в каждый момент времени находящегося в одном состоянии из множества возможных. На вход этому устройству поступают символы одного алфавита, на выходе оно выдаёт символы (в общем случае) другого алфавита.

Конечный автомат — абстрактный автомат без выходного потока, число возможных состояний которого конечно. Результат работы автомата определяется по его конечному состоянию.

Существуют различные варианты задания конечного автомата. Например, конечный автомат может быть задан с помощью пяти параметров: $M = (Q, \Sigma, \delta, q_0, F)$, где:

- Q — конечное *множество состояний* автомата;
- q_0 — *начальное состояние* автомата ($q_0 \in Q$);
- F — множество *заключительных* (или *допускающих*) состояний, таких что $F \subseteq Q$;
- Σ — допустимый *входной алфавит* (конечное множество допустимых входных символов), из которого формируются строки, считываемые автоматом;
- δ — заданное отображение множества $Q \times \Sigma$ во множество $\mathcal{P}(Q)$ подмножеств Q:

  $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ (иногда δ называют *функцией переходов автомата*).

Автомат начинает работу в состоянии q_0, считывая по одному символу входной строки. Считанный символ переводит автомат в новое состояние из Q в соответствии с функцией переходов. Если по завершении считывания входного слова (цепочки символов) автомат оказывается в одном из допускающих состояний, то слово «принимается» автоматом. В этом случае говорят, что оно принадлежит языку данного автомата. В противном случае слово «отвергается».

Диаграмма состояний (или иногда граф переходов) — графическое представление множества состояний и функции переходов. Представляет собой нагруженный однонаправленный граф, вершины которого — состояния КА, дуги — переходы из одного состояния в другое, а нагрузка — символы, при которых осуществляется данный переход. Если переход из состояния q1 в q2 может быть осуществлен при появлении одного из нескольких символов, то над дугой должны быть надписаны все они.

Таблица переходов — табличное представление функции δ. Обычно в такой таблице каждой строке соответствует одно состояние, а столбцу — один допустимый входной символ. В ячейке на пересечении строки и столбца записывается действие, которое должен выполнить автомат, если в ситуации, когда он находился в данном состоянии на входе он получил данный символ.
Детерминированным конечным автоматом (ДКА) называется такой автомат, в котором
для каждой последовательности входных символов существует лишь одно состояние, в
которое автомат может перейти из текущего.
Недетерминированный конечный автомат (НКА) является обобщением детерминированного. Недетерминированность автоматов достигается двумя способами:

1. Существуют переходы, помеченные пустой цепочкой ε
2. Из одного состояния выходит несколько переходов, помеченных одним и тем же символом.

Существует теорема, гласящая, что «Любой недетерминированный конечный автомат может быть преобразован в детерминированный так, чтобы их языки совпадали» (такие автоматы называются эквивалентными).

8. Компиляция и интерпретация. Интерпретация непосредственно и с промежуточной компиляцией. Время связывания. Библиотека времени исполнения.

Компилятор из языка S (исходный язык) в язык T (целевой язык) – это прога, осуществляющая перевод программ с языка S на язык T.
Интерпретатор для языка L – прога, осуществляющая выполнение прог, на яз L.
Виртуальная машина для яз L – сочетание интерпретатора яз L, написанного на яз M, и машины, выполняющей язык M.

В более общем смысле: **Компиляция** - преобразование объектов (данных и операций над ними) с входного языка в объекты на другом языке для всей программы в целом. Задача выполнения не стоит. **Интерпретация** - анализ отдельного объекта на входном языке с одновременным выполнением (интерпретацией).

Типы интерпретаторов:

Простой интерпретатор анализирует и тут же выполняет программу покомандно, по мере поступления её исходного кода на вход интерпретатора. Достоинством такого подхода является мгновенная реакция. Недостаток — такой интерпретатор обнаруживает ошибки в тексте программы только при попытке выполнения команды с ошибкой.

Интерпретатор с промежуточной компиляцией.— это система из компилятора, переводящего исходный код программы в промежуточное представление, и собственно интерпретатора, который выполняет полученный промежуточный код (виртуальная машина). Достоинством таких систем является большее быстродействие выполнения программ. Недостатки — большее время запуска.

Достоинства и недостатки интерпретаторов: + :

1) Бóльшая переносимость интерпретируемых программ — программа будет работать на любой платформе, на которой есть соответствующий интерпретатор.2) Меньшие размеры кода по сравнению с машинным кодом, полученным после обычных компиляторов.

-: 1) Интерпретируемая программа не может выполняться отдельно без программы-интерпретатора. 2) Интерпретируемая программа выполняется медленнее, поскольку промежуточный анализ исходного кода и планирование его выполнения требуют дополнительного времени. 3) Практически отсутствует оптимизация кода (для простых), что приводит к дополнительным потерям в скорости работы интерпретируемых программ.

Связывание - процесс установления соответствия между объектами и их свойствами в программе на формальном языке (операции, операторы, данные) и элементами архитектуры компьютера (команды, адреса).

Временем связывания называется соответственно фаза подготовки программы к выполнению (трансляция, компоновка, загрузка), на которой производится это действие. Заметим, что различные характеристики одного и того же объекта (например, переменной) могут связываться с различными элементами архитектуры в разное время, то есть процесс связывания не является одномоментным.

Классификация времени связывания

1. *Во время выполнения программы*. Связывание часто происходит во время выполнения программы. Сюда входят связывание переменных с их значениями, а также (во многих языках) связывание переменных с конкретными областями памяти. Подкатегории:
    - *При входе в подпрограмму или блок*. В нек. языках связывание формальных параметров с фактическими и связывание формальных параметров с определенными областями памяти происходит только во время входа в подпрограмму.
    - *В произвольных точках программы во время выполнения*. Пример – присв-е.
2. *Во время трансляции* (*компиляции*).
    - *Связывание по выбору программиста*.
    - *Связывание по выбору транслятора*.
    - *Связывание по выбору загрузчика*. Это происходит во *время загрузки*, также называемое *временем редактирования связей*.
3. *Время реализации языка*. Некоторые аспекты определений языка в рамках некоторой его реализации могут быть одними и теми же для всех выполняемых программ, однако они могут различаться в других его реализациях.
4. *Время определения языка*.

Также понятия связывание имеет отношение к полиморфизму и виртуальным функциям. При вызове виртуальной функции для объекта базового класса программа должна идентифицировать, к какому конкретно классу относится текущий объект, и выбрать соответствующую ему функцию. Таким образом, для виртуального метода осуществляется и возможно только **позднее связывание**, т.е. привязка осуществляется на этапе выполнения. (**Раннее связывание** - привязка действия к объекту осуществляется на этапе компиляции.)

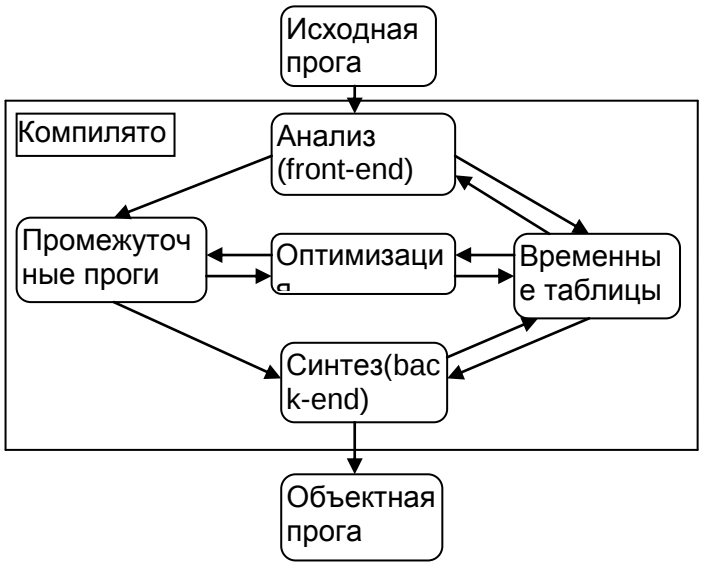
Библиотека времени выполнения – специальная библиотека, используемая компилятором, для реализаций встроенных функция языка во время выполнения программы. Она практически всегда включает функции для ввода-вывода и управления памятью.

В RTL также находятся такие специальные функции как, например, проверка на выход за границы массива, динамическая проверка типов, генерация исключений, и т.д.

RTL осуществляет доступ программ к системным вызовам. RTL всегда зависит от компилятора и/или от платформы.

Из лекций Макарова: **Библиотека времени исполнения (Common Language Runtime, CLR)**, функционирует как на этапе выполнения, так и на этапе разработки. Во время выполнения кода она следит за динамикой многопотоковых приложений, обеспечивает взаимосвязь процессов, поддерживает их безопасность, автоматизирует процедуры выделения и освобождения памяти. На этапе разработки CLR автоматизирует типовые задачи, решаемые программистом, значительно упрощая использование новых технологий.

9. Компиляторы: архитектура, фазы компиляции, методы синтаксического анализа.



Стадии компиляции: анализ (не зависит от целевого яз, тут м пораждаться сообщения об ошибках), оптимизация (зависит только от промежуточного представления проги), синтез (почти не зависит от исх яз). Если компиляция проги продолжается после обнаружения в ней ошибки, компилятор выполняет **восстановление при ошибках**.

Стадии компиляции – сложные транслирующие преобразования => их рассматривают как композиции более простых преобразований – фаз компилляции.

Фазы анализа:

1. **чтение входного потока** – фаза компил, осуществляющая преобразование образа текста в последовательность кодовых точек во внутреннем для компил. стандарте кодирования текста.
2. **лексический анализ** (линейный анализ сканирование) – //-//-, объединяющая последовательно идущие во вход потоке КТ в группы – лексемы исх яз.
3. **синтаксический анализ** (иерархический анализ, разбор) – //-//-, группирующая лексемы, пораждаемые на фазе лексич анализа, в синтаксич структуры. Широко используются генераторы лексич и синтаксич анализаторов (lex и yacc).
4. **семантический анализ** – //-//-, выполняющая проверку синтаксического дерева на соответствие его компонентов контекстным ограничениям (правилам видимости идентификаторов, проверка типов и т.д.). Тут копится инфа о типах для генерации кода.
5. **генерация промежуточного представления** – //-//-, выполняющая перевод синтаксического дерева в форму удобную для послед оптимизации и генерации кода.

Фазы оптимизации сильно зав от промежуточного представления.

Фазы синтеза:

1. **распределение памяти** – фаза компиляции, связанная с переносом структур данных, определенных в промежуточном представлении проги, в модель целевого языка.
2. **генерация кода на целевом языке** – //-//-, выполняющая перевод проги из промежуточного представления в целевой язык.
3. **оптимизация**, зависящая от целевого яз (постобработка) – //-//-, связанная с оптимизацией объектной проги, полученной в результате генерации кода.

Фазы комп м располагаться **последовательно или параллельно**, возможны сложные сочетания послед и парал выполнения. **Проход** – выполнение группы параллельно работающих фаз. Проход м б реализован как отдельная прога. Параллелизм фаз в рамках прохода означает, что эти фазы работают в режиме обменивающихся данными сопрограмм (т.е. реального парралелизма нет и фазы выполняются поочередно). Если при компиляции происходи последовательное выполнение n-проходов, то **компиляция n-проходная**.

Задача синтаксического анализа. Дано: G = (N,T,S,P) – КС-гр языка, x из T* - цепочка. Требуется определить. Является ли цепочка x предложением в грамматике G. Если да, то построить дерево вывода цепочки x в G.

Методы:

Алгоритм предсказывающего синтаксического разбора. Последовательность формирования дерева вывода – **сверху-вниз**. Основная проблема – определить правило вывода, которое нужно применить к терминалу. Таблица анализатора задает функцию δ: N×T -> (NUT)*U{ERROR}. На выходе – последовательность правил вывода.

Алгоритм Эрли подходит для любой КС-гр и для заданного предложения языка позволяет найти все левые выводы. Алгоритм Эрли работает «сверху-вниз». **Работа алгоритма** основана на построении последовательности множеств называемых множествами Эрли. Для входной цепочка a1a2…an конструируются n+1 множеств: начальное множество S0 и по одному множеству Si для каждого входного символа ai. Элемент мн-ва Эрли, принадлежащий мн-ву Sk – это тройка (p,i,j) в которой: p – правило грамматики, i – позиция в правой части правила, показывающая какая часть правила уже обработана, j<=k – мн-ва Эрли, в котором началось распознавание по правилу p. В процессе работы алгоритма одни элементы порождают другие. Для того чтобы м было восстановить последовательность применяемых правил, необходимо строить дерево потомков, в узлах которого находятся элементы Эрли, а дуги отражают тот факт, что один элемент породил другой элемент.

Рекурсивный спуск. Рассмотрим метод рекурсивного спуска применительно к LL(1)-гр, записанным в РБНФ. В синтаксическом анализаторе, написанном методом рекурсивного спуска, каждому нетерминалу грамматики

соответствует отдельная функция, в которой закодирован эффект применения соответствующего нетерминалу правила (будем считать правило единственно). **Цель этой функции** – анализ последовательности токенов, которые по ее запросу выдает лексический анализатор, и проверка соответствия этой последовательности правилу грамматики:

```
X() {       PARSE();      }
```

Синтаксический разбор типа «перенос-свертка». Последовательность формирования дерева вывода в процессе синтаксического разбора снизу-вверх. ПС-алгоритм строит правосторонний вывод предложения языка в обратном порядке. ПС-алгоритм работает с двумя структурами данных: **стек** (помещаются символы грамматики (NUT), дно помечено $) и **входной буфер** (содержит нераспознанный суффикс входной цепочки, оканчивается $). ПС-алгоритм выполняет 4 действия:

1. перенос – символ в начале входного буфера переносится в стек
2. свертка – основа на вершине стека заменяется нетерминалом
3. допуск – успешное окончание разбора (во входн буфере $, в стеке - $S, S – аксиома гр.)
4. ошибка – вызов подпрограммы реакции на ошибку во входном потоке

Основные проблемы – поиск основы в стеке и выбор правила, по которому н осуществлять свертку. В общем случае ПС-алгоритм осуществляет возвраты.

Алгоритм LR(1)-анализа является вариантом ПС-алгоритма, работающим за линейное время. В стеке LR(1)-анализатора содержится последовательность состояний, через которые прошел SLR-распознаватель (или более сложный LR-распознаватель) в процессе распознавания активного префикса. LR(1)-анализатор управляется таблицами распознавателя, не содержащими конфликтов.

10. Фундаментальные структуры данных в программировании

Массив – это коллекция данных, которая в большинстве языков программирования имеет фиксированную длину и состоит из элементов одного типа.

+: 1) Массивы помогают определять множества данных в осмысленные группы.

2) Имена массивов с индексами минимизируют потребность в слежении за многими элементами данных с различными именами.
3) Использование индексов обеспечивает непосредственный и автоматический доступ к любому элементу в массиве.
4) Индексация позволяет так же проводить с помощью циклов DO и FOR автоматическую, быструю и эффективную обработку всех данных или выделенных подмножеств данных, хранимых в массивах.

-: лучше всего массивы годятся для данных, порядок которых не изменяется. Если порядок элементов в массиве подвергается изменению, то каждый раз, когда порядок меняется, перестановка элементов требует очень много времени.

В **связном списке** элементы линейно упорядочены, но порядок определяется не номерами, как в массиве, а указателями, входящими в состав элементов списка. Списки являются удобным способом хранения динамических множеств. *Линейный однонаправленный список* – это конечный набор пар, каждая из которых состоит из информационной части INFO и указывающей части LINK (указывает на следующий элемент списка). *Двусвязный список (Двунаправленный связанный список)* По двусвязному списку можно передвигаться в любом направлении — как к началу, так и к концу. В этом списке проще производить удаление и перестановку элементов, т.к. всегда известны адреса тех элементов списка, указатели которых направлены на изменяемый элемент. *Кольцевой связный список.* Последний элемент кольцевого списка содержит указатель на первый, а первый (в случае двусвязного списка) — на последний.

Ассоциативный массив (словарь) — абстрактный тип данных (интерфейс к хранилищу данных), позволяющий хранить пары (ключ, значение) и поддерживающий операции добавления пары, а также поиска и удаления пары по ключу: • INSERT(ключ, значение) • FIND(ключ) • REMOVE(ключ). Предполагается, что ассоциативный массив не может хранить две пары с одинаковыми ключами. Используется в скриптовых языках (JavaScript, Ruby, PHP).

Хеш-табли́ца — это структура данных, реализующая интерфейс ассоциативного массива, она позволяет хранить пары (ключ, значение) и выполнять три операции: добавление новой пары, поиск и удаление пары по ключу. Выполнение операции в хеш-таблице начинается с вычисления хеш-функции от ключа. Получающееся хеш-значение i = hash(key) играет роль индекса в массиве H. Затем выполняемая операция (добавление, удаление или поиск) перенаправляется объекту, который хранится в соответствующей ячейке массива H[i]. Если все ключи элементов известны заранее (или очень редко меняются), то для них можно найти некоторую совершенную хеш-функцию. Такие хеш-таблицы не нуждаются в механизме разрешения коллизий (когда для различных ключей получается одно и то же хеш-значение), и называются **хеш-таблицами с прямой адресацией.** Число хранимых элементов, делённое на размер массива H (число возможных значений хеш-функции), называется коэффициентом заполнения хеш-таблицы, от него зависит среднее время выполнения операций.

Стек(LIFO) – структура данных, для которой определены 2 операции (push и pop). Они позволяют добавлять и удалять элементы с верхушки стека. Для отслеживания точек возврата из подпрограмм используется стек вызовов. Языки программирования высокого уровня также используют стек вызовов для передачи параметров при вызове процедур.

Очередь (FIFO) - Для этой структуры данных вводится 2 индекса: «голова» - Head и «хвост» - Tail. Элементы, добавляемые в очередь, оказываются в ее хвосте, а элементы, удаляемые из очереди, находятся у нее в голове.

Дерево – это связный граф (между любой парой вершин существует по крайней мере один путь), не содержащий циклов (ациклический граф). Ацикличность означает, что в дереве существует только по одному пути между парами вершин. **Ориентированное (направленное) дерево** —только одна вершина имеет нулевую степень захода (корень), а все остальные вершины имеют степень захода 1 (в них ведёт ровно по одной дуге – вершины и листья). Дерево определяется как конечное множество T одного или более узлов со следующими свойствами:

1. существует один корень дерева T

2. остальные узлы (за исключением корня) распределены среди непересекающихся множеств T1,...,Tm, и каждое из множеств является деревом; деревья T1,...,Tm называются поддеревьями данного корня T.

B-дерево – сбалансированное дерево –длина пути от корня к любому листу одна и та же (используется в индексах).

11. Графы и деревья: основные понятия, алгоритмы нахождения кратчайших путей.

Граф или неориентированный граф G — это упорядоченная пара G: = (V,E), для которой выполнены следующие условия: V это множество вершин или узлов, E это множество пар вершин, называемых рёбрами. Вершины и рёбра графа называются также элементами графа, число вершин в графе | V | — порядком, число рёбер | E | — размером графа.

Ориентированный граф (сокращённо орграф) G — это упорядоченная пара G: = (V,A), для которой выполнены следующие условия: V это множество вершин или узлов, A это множество (упорядоченных) пар различных вершин, называемых дугами или ориентированными рёбрами.

Дуга — это упорядоченная пара вершин (v, w), где вершину v называют началом, а w — концом дуги. Можно сказать, что дуга v w ведёт от вершины v к вершине w.

Смешанный граф G — это граф, в котором некоторые рёбра могут быть ориентированными, а некоторые — неориентированными. Записывается упорядоченной тройкой G: = (V,E,A), где V, E и A определены так же, как выше.

Путём (или цепью) в графе называют конечную последовательность вершин, в которой каждая вершина (кроме последней) соединена со следующей в последовательности вершин ребром. Ориентированным путём в орграфе называют конечную последовательность вершин vi , для которой все пары (vi,vi + 1) являются ориентированными рёбрами.

Циклом называют путь, в котором первая и последняя вершины совпадают. При этом **длиной** пути (или цикла) называют число составляющих его рёбер. Путь (или цикл) называют **простым**, если ребра в нём не повторяются; **элементарным**, если он простой и вершины в нём не повторяются. Отношение «существует путь из u в v» (отношение эквивалентности) разбивает это множество на классы эквивалентности - компоненты связности графа. Одна компонента => **граф связный**.

Дерево – связный ацикличный граф. **Полный граф** – любые 2 вершины соединены ребром. **Двудольный граф** - его вершины можно разбить на два непересекающихся подмножества V1 и V2 так, что всякое ребро соединяет вершину из V1 с вершиной из V2.

Алгоритм Дейкстры - находит кратчайшее расстояние от одной из вершин графа до всех остальных. Алгоритм работает только для графов без рёбер отрицательного веса.

Каждой вершине из V сопоставим метку — минимальное известное расстояние от этой вершины до a. Алгоритм работает пошагово — на каждом шаге он «посещает» одну вершину и пытается уменьшать метки. Работа алгоритма завершается, когда все вершины посещены.

Инициализация. Метка самой вершины a полагается равной 0, метки остальных вершин — бесконечности. Это отражает то, что расстояния от a до других вершин пока неизвестны. Все вершины графа помечаются как непосещённые.

Шаг алгоритма. Если все вершины посещены, алгоритм завершается. В противном случае, из ещё не посещённых вершин выбирается вершина u, имеющая минимальную метку. Для каждого соседа вершины u, кроме отмеченных как посещённые, рассмотрим новую длину пути, равную сумме значений текущей метки u и длины ребра, соединяющего u с этим соседом. Если полученное значение длины меньше значения метки соседа, заменим значение метки полученным значением длины. Рассмотрев всех соседей, пометим вершину u как посещенную и повторим шаг алгоритма.

Алгоритм Беллмана — Форда — поиск кратчайшего пути во взвешенном графе. За время O(|V| × |E|) алгоритм находит кратчайшие пути от одной вершины графа до всех остальных. Допускает рёбра с отрицательным весом. Заметим, что кратчайших путей может не существовать. Так, в графе, содержащем цикл с отрицательным суммарным весом, существует сколь угодно короткий путь от одной вершины этого цикла до другой (каждый обход цикла уменьшает длину пути). Цикл, сумма весов рёбер которого отрицательна, называется отрицательным циклом. Начиная с некоторой вершины, просматриваем по «доступным» ребрам соседние вершины, и пытаемся улучшить до них расстояние в массиве dist[..] и сделать как можно меньше. Этот процесс называется «релаксация». Если «нащупали» (по ребрам) такие вершины, то обновляем им расстояния и заносим их вершины в очередь «попробовать из них улучшить граф». Если придется зайти в одну вершину дважды, то сделаем это. Но вот вопрос, готовы ли мы к тому, что будут попадаться «отрицательные циклы», по которым можно вечно крутиться, уменьшая расстояние до вершин? Процесс не закончится. Поэтому, «радиус осмотра» вершин ограничим числом N (числом самих вершин). Этого будет гарантированно достаточно для того, чтобы просчитать минимальное расстояние до любой вершины, а главное алгоритм в любом случае завершится. Алгоритм Беллмана — Форда позволяет очень просто определить, существует ли в графе G отрицательный цикл, достижимый из вершины s. Достаточно произвести внешнюю итерацию цикла не |V| - 1, а ровно |V| раз. Если при исполнении последней итерации длина кратчайшего пути до какой-либо вершины строго уменьшилась, то в графе есть отрицательный цикл, достижимый из s.

12. Реляционная теория. Отношения. Реляционная алгебра. Теория нормализации. Реляционное исчисление. Целостность. Определение реляционной модели.

Реляционная теория (или реляционная модель) – модель данных, описывающая структуру, операции и специальные правила для организации хранения и обработки данных. Реляционная модель состоит из трёх частей: структурной, манипуляционной и целостной.

Структурная часть говорит, что единственной структурой данных, используемой в реляционных базах данных, является нормализованное n-арное отношение.

Отношение – это подмножество доменов. Домен – это некоторое множество элементов. Кортеж представляет собой последовательность из n-элементов, где элементы соответствуют доменам в отношении. Таким образом отношение – это множество кортежей, соответствующих доменам (заголовкам таблицы).

Манипуляционная часть утверждает два механизма манипулирования данными – реляционная алгебра и реляционное исчисление.

Реляционная алгебра представляет собой набор операций, которые можно совершать над отношениями. Отдельно стоит заметить, что результатом всех операций с отношениями являются отношения (т.е. реляционная алгебра – как и реляционное исчисление – замкнута на понятии отношения.)

Основные операции реляционной алгебры:

1. объединение отношений. (В исходное отношение попадают кортежи, принадлежащие хотя бы одному из объединяемых отношений. Т.е. если в отношении А были кортежи a1, a2, а в кортеже В были кортежи b1, b2, a1, то в отношение R, представляющее объединение А и В, попадут кортежи a1, a2, b1, b2, a1 – заметьте, что кортеж a1 попадёт в отношение дважды.)
2. пересечение отношений. (В исходное отношение попадают кортежи, принадлежащие обоим отношениям. Т.е. А пересечение В даст {a1}.)
3. взятие разности отношений. (В исходное отношение попадают кортежи, которые есть в первом отношении, кроме тех, которые есть и во втором отношении. Т.е. А минус В даст {a2}.)
4. прямое произведение отношений.

Специальный реляционный операции:

1. ограничение отношения. (WHERE)
2. проекция отношения. (SELECT)
3. соединение отношений. (Аналог оператору JOIN в SQL. Результатом соединения двух отношений является конкатенация кортежей этих отношений, удовлетворяющих некоторому условию.)
4. деление отношений. (У операции реляционного деления два операнда — бинарное и унарное отношения. Результирующее отношение состоит из одноатрибутных кортежей, включающих значения первого атрибута кортежей первого операнда таких, что множество значений второго атрибута (при фиксированном значении первого атрибута) совпадает со множеством значений второго операнда.)

Целостная часть говорит о двух требованиях целостности: целостность сущностей и целостность по ссылкам. Целостность сущностей подразумевает уникальность всех кортежей в отношении (т.е. нет кортежей со всеми одинаковыми элементами); целостность по ссылкам требует, чтобы кортежи, на которые ссылаются (из других отношений или же из самого отношения), существовали.

Ключ - это значение некоторого атрибута или атрибутов в кортеже отношения, который представляет экземпляр сущности в реляционной модели данных.

Нормальная форма — свойство отношения в реляционной модели данных, характеризующее его с точки зрения избыточности, которая потенциально может привести к логически ошибочным результатам выборки или изменения данных.

1: Т не имеет повторяющихся записей, отсутствуют повторяющиеся группы полей, столбцы не упорядочены, строки не упорядочены.

2: Т нах в 1 НФ, любое неключевое поле однозначно идентифицируется полным набором ключевых полей.

3: Т нах в 2 НФ, ни одно неключ. поле не идентифицируется с помощью другого неключ. поля.

3 опр (Бойса-Кода): Т нах в 3 НФ, отсутствует зависимость полей составного ключа от неключ. полей.

4: Т нах. в БКНФ, существует многомерная (многозначная) зависимость *A* от *B*, а все остальные атрибуты отношения *R* однозначно зависят от атрибута *A* (где *A* и *B* — это атрибуты отношения *R*).

5: Т нах в 4 НФ, любая зависимость в таблице является зависимостью только от первичного ключа.

13. Реляционные базы данных. Проектирование реляционных баз данных. Инфологическое проектирование. Построение ER-моделей и их преобразование в структуру баз данных. Проектирование физической структуры базы данных. Индексирование.

Реляционные базы данных – это базы данных, основанных на реляционной модели. Данные в реляционной базе данных (РСУБД) хранятся в таблицах. Таблицы (аналог отношения) состоят из столбцов и строк, на пересечении которых находятся данные. Запросы к таблицам возвращают таблицы, которые также можно использовать в запросах.

При проектировании баз данных решаются две проблемы:

1. Каким образом отобразить объекты предметной области в абстрактные объекты модели данных, чтобы это отображение не противоречило семантике предметной области и было по возможности лучшим (эффективным, удобным и т.д.)? (Проблема логического проектирования.)
2. Как обеспечить эффективность выполнения запросов к базе данных, т.е. каким образом, имея в виду особенности конкретной СУБД, расположить данные во внешней памяти, создание каких дополнительных структур (индексов) потребовать и т.д.? (Проблема физического проектирования.)

Инфологическое проектирование сводится к построению инфологической модели. Для этого собирается информация о предметной области (это делает системный аналитик). Системный аналитик определяет область применения базы данных, анализирует информационные потоки на предмет совместного использования данных. В результате получается схема информационных потоков.

ER-модель представляет из себя набор ER-диаграмм.

ER (Entity-Relation) диаграмма позволяет отображать объекты базы данных и связи между ними. Основные понятия ER-модели – это сущность, связь и атрибут.

Сущность – это, фактически, таблица. Связь – это графически отображаемая связь между двумя сущностями. Атрибуты – это некоторая дополнительная информация о сущности. Т.е. атрибуты – это столбцы таблицы (сущность – это название таблицы, а атрибуты – это конкретные столбцы.)

Также у ER-модели есть несколько дополнительных элементов: подтипы и супертипы (по аналогии с ООП), связи «много ко многим», уточняемые степени связи, домены.

Получение реляционной схемы из ER-схемы:

1. Каждая простая сущность (сущность, не имеющая подтипов и не являющаяся подтипом) преобразовывается в таблицу. Имя сущности становится именем таблицы.
2. Каждый атрибут становится столбцом таблицы.
3. Компоненты уникального идентификатора сущности делаются первичным ключом.
4. Связи много-к-одному и один-к-одному становятся внешними ключами.
5. Создаются индексы для первичного ключа, внешних ключей и для атрибутов, по которым чаще всего производится выборка.
6. Если в схеме присутствуют подтипы, то можно либо для каждого подтипа создавать отдельную таблицу, либо сделать одну таблицу для всех подтипов.
7. Для связей много-ко-многим создаются дополнительные таблицы.

Физическое проектирование занимается вопросами эффективности выполнения запросов. Фактически, это означает выяснение того, какие индексы нужно сделать, какого оптимального размера должны быть записи в таблицах (на основе знания внутреннего устройства СУБД).

Индекс – это некоторая структура данных, позволяющая осуществлять поиск по некоторому условию эффективно. Если в поиске, напр., по имени не использовать индексов, то поиск будет производиться путём полного перебора. Зачастую для индексов используют структуру данных B-дерево (см. графы).

Поиск по B-дереву представляет собой прохождение от корня к листьям. Например, для строковых значений дерево может состоять из букв значений: первый уровень – а, б, в, г.... я, следующий уровень – это вторые буквы и т.д. Таким образом, чтобы найти слово «рыба» нужно найти на первом уровне дерева вершину с буквой «р», если такой не оказалось, значит значение не найдено, если вершина найдена, то среди её потомков нужно найти букву «ы» и т.д.

Также для индексов используются хэш-таблицы. Работа хэш-таблицы основана на некоторой хэш-функции, которая принимает в себя значение, а на выходе даёт целое положительное число. Таким образом, чтобы найти строку, нужно вычислить с помощью хэш-функции значение, потом посмотреть, есть ли такое значение.

14. Теория распределенных систем К.Дейта. Распределенные базы данных.

Впервые задача об исследовании основ и принципов создания и функционирования распределенных информационных систем была поставлена известным специалистом в области баз данных К. Дейтом
Собственно в основе распределенных АИС лежат две основные идеи:
• много организационно и физически распределенных пользователей, одновременно работающих с общими данными — общей базой данных (пользователи с разными именами, в том числе располагающимися на различных вычислительных установках, с различными полномочиями и задачами);
• логически и физически распределенные данные, составляющие и образующие тем не менее единое взаимосогласованное целое — общую базу данных (отдельные таблицы, записи и даже поля могут располагаться на различных вычислительных установках или входить в различные локальные базы данных).
Крис Дейт сформулировал также основные принципы создания и функционирования распределенных баз данных. К их числу относятся:
• прозрачность расположения данных для пользователя (для пользователя распределенная база данных должна представляться и выглядеть точно так же, как и нераспределенная);
• изолированность пользователей друг от друга (пользователь должен «не чувствовать», «не видеть» работу других пользователей в тот момент, когда он изменяет, обновляет, удаляет данные);
• синхронизация и согласованность (непротиворечивость) состояния данных в любой момент времени.

Основная задача систем управления распределенными БД (РБД) состоит в обеспечении средства интеграции локальных БД, располагающихся в некоторых узлах вычислительной сети, с тем, чтобы пользователь, работающий в любом узле сети, имел доступ ко всем этим БД как к единой.

При этом должны обеспечиваться:

- простота использования системы;
- возможности автономного функционирования при нарушениях связности сети или при административных потребностях;
- высокая степень эффективности.

РБД могут быть однородными (когда используется одна и та же СУБД для каждой из удалённых БД; например, MS SQL Server) и неоднородными. Создание неоднородной СУБД задача нетривиальная. Самая большая проблема – низкая производительность неоднородной РБД.

Для того, чтобы работала РБД, необходимо обеспечить правильную параллельную работу с БД. Транзакции в РБД называются распределёнными. Как правило, транзакция начинается в главном узле при обращении к какой-либо секции ранее подготовленного (на этапе компиляции) модуля доступа. Модуль доступа загружается в виртуальную память задачи, обращение к секции модуля доступа — это вызов подпрограммы. Выполнение одной транзакции, инициированной в некотором узле сети А влечет инициирование транзакций в дополнительных узлах.

Основной проблемой является проблема согласованного завершения распределенной транзакции, чтобы результаты ее выполнения во всех затронутых ею узлах были либо отображены в состояние локальных БД, либо полностью отсутствовали. Для достижения этой цели используется двухфазный протокол завершения распределенной транзакции: имеется ряд независимых транзакций-участников распределенной транзакции, выполняющихся под управлением транзакции-координатора. Решение об окончании распределенной транзакции принимается координатором. После этого выполняется первая фаза завершения транзакции, когда координатор передает каждому из участников сообщение «подготовиться к завершению». Получив такое сообщение, каждый участник переходит в состояние готовности как к немедленному завершению транзакции, так и к ее откату. После этого каждый участник, успешно выполнивший подготовительные действия, посылает координатору сообщение "готов к завершению". Если координатор получает такие сообщения ото всех участников, то он начинает вторую фазу завершения, рассылая всем участникам сообщение «завершить транзакцию», и это считается завершением распределенной транзакции. Если не все участники успешно выполнили первую фазу, то координатор рассылает всем участникам сообщение «откатить транзакцию», и тогда эффект воздействия распределенной транзакции на состояние БД отсутствует. Координатор - транзакция, выполняющаяся в главном узле, т.е. та, по инициативе которой возникли дополнительные транзакции.

15. Базы знаний. Принципы разработки интеллектуальных систем.

База знаний предназначена для хранения экспертных знаний о предметной области, используемых при решении задач ЭС. Содержит факты (или утверждения) и правила (модель предметной области).

Факты представляют собой краткосрочную информацию в том отношении, что они могут изменяться, например, в ходе консультации.

Правила представляют более долговременную информацию о том, как порождать новые факты или гипотезы из того, что сейчас известно. База знаний обладает творческими возможностями, она активно пытается пополнить недостающую информацию. Но следует помнить, что эти знания не воплощены в какую-то программу, а представляют собой данные для высокоуровневого интерпретатора, а именно - для машины вывода.

Машина вывода - механизм, который необходим для построения логических вычислений (механизм рассуждений, оперирующий знаниями и данными с целью получения новых данных из знаний и других данных, имеющихся в рабочей памяти). Для этого обычно используется программно реализованный механизм дедуктивного логического вывода (какая- либо его разновидность) или механизм поиска решения в сети фреймов или семантической сети.

Существует 2 режима: **прямая** цепочка рассуждений (использование фактов) и **обратная** цепочка рассуждений (подтверждение или опровержение фактов).

Наиболее удачные системы используют комбинации этих способов рассуждений.

В базе знаний с помощью выбранной модели представления знаний хранятся знания эксперта о предметной области, способы анализа поступающих фактов и методы вывода, т.е. новые знания, порожденные на основании имеющихся и вновь поступающих. Представленные знания зависят от специфики предметной области. Они могут быть качественные и количественные, точные и приближенные, конкретные и общие, алгоритмические и эвристические.

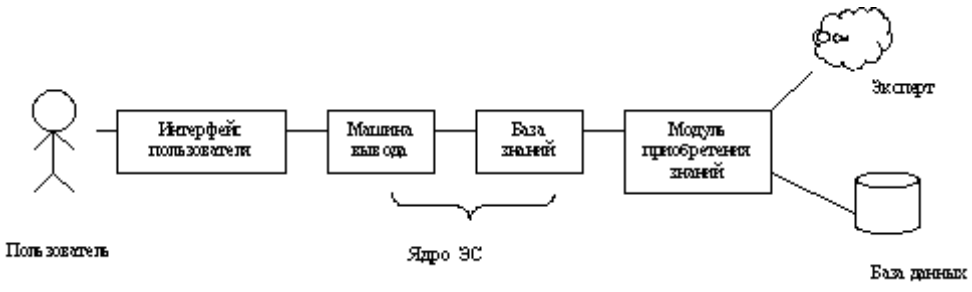
Для представления знаний используются следующие модели:

• Логические – базируются на представлении знаний в системе логики предикатов первого порядка. Правила формальной логики постепенно расширяются, приближаясь к человеческой, которая характеризуется нечеткостью. В связи с чем появились модальная, многозначная, немонотонная, псевдофизическая и другие виды логик.

• Продукционные – представляют знания в форме предикатов первого порядка, а правила манипулирования ими – с помощью конструкций ЕСЛИ – ТО. База знаний состоит из правил типа: «**ЕСЛИ** рентабельность снизилась **И** прибыль увеличилась **ТО** себестоимость продукции выросла»

• Фреймовые – отражают систематизированную в виде единой теории модель памяти человека. Основной элемент модели – фрейм – отражает структуру данных для описания понятийных структур классов, объектов. Все фреймы взаимосвязаны и образуют единую систему, в которой объединены факты (описательные знания), правила (содержащие И, ИЛИ) и процедуры их обработки. С помощью фреймов можно определить объекты с наследуемыми свойствами и конструкции структуры более общего характера с большим числом возможностей.

• Семантическая сеть – наиболее удобная, наглядная и понятная экспертам модель представления знания. Под семантической сетью, как правило, подразумевается граф, узлы которого соответствуют понятиям или объектам, а дуги отражают отношения между ними.



ИС представляет собой человеко-машинную систему, которая в простейшей форме может быть описана так:

Основу ИС представляет база знаний и данных, содержащая модель предметной области. Различные ИС отличаются средствами представления знаний.

Часть базы знаний содержит фундаментальные знания и данные о предметной области. Различные ИС отличаются средствами представления знания. Эти знания могут изменяться и пополняться. Например, для медицинской ИС такими знаниями будут знания о болезнях, методах их лечения, лекарственных средствах. Другая часть БЗ содержит знания, которые со временем изменяются более интенсивно: данные и знания о больных, ходе лечения, применяемых средствах лечения и т.п.

Для ответа на поставленные пользователем вопросы требуется система извлечения ответов, называемая машиной логического вывода или интерпретатором.

ИС кроме ответа на предъявляемые пользователем вопросы должна объяснять причины, по которым дается тот или иной ответ, то или иное решение. Объяснение осуществляется соответствующей системой пояснения решений.

Взаимодействие пользователя с ИС осуществляется посредством оболочки, представляющей пользовательский интерфейс для двух категорий разработчиков ИС – программистов и для инженеров по

знаниям. Программисты осуществляют разработку и тестирование ИС как программы, разработанной на том или ином языке программирования. Инженеры по знаниям осуществляют взаимодействие между экспертами и ИС, пополняя и корректируя базу знаний.

ИС отличаются от других программ ИИ своей целью и построением. Под целью мы понимаем задачи, которые подходят для человека-эксперта, а под построением – создание не просто механической, а интеллектуальной программы. Критериями оценки ИС являются следующие:

- отражает ли внутренне функционирование программы подход к проблеме со стороны человека;
- может ли программа объяснять свои действия способом, понятным человеку;
- может ли программа взаимодействовать с оператором посредством гибкого диалога подобного диалогу на естественном языке.

ИС используют все методы программирования, которые применяются в других программах ИИ. Некоторые программы занимают пограничное положение между ИС и прогаммой ИИ. Программы игры в шахматы – программы грубой силы, т.е. они заносят в каталог все возможные для компьютера ходы и, кроме того, учитывают все возможные контрходы игрока-человека, сделанные на ход машины, а также все приемлемые ходы компьютера в ответ на ход человека игрока.

16. Языки логического программирования: исчисление предикатов, метод резолюций, декларативная семантика.

Логи́ческое программи́рование — парадигма программирования, а также раздел дискретной математики изучающий методы и возможности этой парадигмы, основанная на выводе новых фактов из данных фактов согласно заданным логическим правилам. Логическое программирование основано на теории математической логики. Самым известным языком логического программирования является Prolog, являющийся по своей сути универсальной машиной вывода, работающей в предположении замкнутости мира фактов. Пролог (Prolog, программирование в логике) - один из наиболее широко используемых языков логического программирования. Как и для других декларативных языков, при работе с ним мы описываем ситуацию (правила и факты) и формулируем цель (запрос), позволяя интерпретатору Пролога найти решение задачи за нас.

У каждого из языков программирования есть свой круг задач, при решении которых он используется с наибольшей эффективностью. Для Пролога это задачи, связанные с разработкой систем искусственного интеллекта (различные экспертные системы, программы-переводчики, интеллектуальные игры). Он используется для обработки естественного языка и обладает мощными средствами, позволяющими извлекать информацию из баз данных, причем методы поиска, используемые в нем, принципиально отличаются от традиционных.

Декларативные формы представления знаний - знания представленные в виде утверждений, описывающих объекты, их свойства, но не дают инструкций, как решать задачу. Они используются в продукционных, редукционных, и логических языках (Пролог, Lisp).

Языки предназначенные для описания предметных областей называются языками представления знаний. Универсальным языком представления знаний является естественный язык, но применение естественного языка для машинного представления знаний наталкивается на ряд препятствий, главным из которых является отсутствие формальной семантики естественного языка. Семантика – это смысловое значение единиц языка.

Декларативная семантика. В Прологе обычно применяются две семантические модели: декларативная и процедурная. Семантические модели предназначены для объяснения смысла программы. В декларативной модели рассматриваются отношения, определенные в программе. Для этой модели порядок следования предложений в программе и условий в правиле не важен.

Процедурная модель рассматривает правила как последовательность шагов, которые необходимо успешно выполнить для того, чтобы соблюдалось отношение, приведенное в заголовке правила.

Исчисле́ние выска́зываний — это формальная теория, в которой осуществляется попытка формализации понятий логического закона и логического следования.

Исчисление предикатов является обобщением исчисления высказываний, позволяющим использовать параметры (называемые также аргументами или переменными) в высказываниях.

Правило резолюции.

Из истинности двух дизъюнкций, одна из которых содержит дизъюнкт, а другая – его отрицание, следует (выводима) формула, являющаяся дизъюнкцией исходных формул без упомянутого дизъюнкта и его отрицания. $\alpha \vee \beta, \neg \beta \cup \gamma \mapsto \alpha \cup \gamma$ или $\neg \alpha \supset \beta, \beta \supset \gamma \mapsto \neg \alpha \supset \gamma$

Метод резолюции (обратного вывода) заключается в следующем: в базе данных вводится правило, которое содержит отрицание цели, и оно проверяется, пока не будет найдено противоречие (т.е. правило верно) или оно будет подтверждено.

В обратном выводе все делается следующим образом: на первом шаге подбирается правило вывода, следствием которого является целевая формула, а условие может быть образовано из формул начальной базы знаний с использованием, если это необходимо, унификации. Унифицированные формулы, входящие в условие, принимаются за новые целевые формулы, обычно называемые подцелевыми формулами. И все повторяется сначала, но уже для каждой из них, и так до тех пор, пока все унифицированные подцелевые формулы не окажутся аксиомами, входящими в начальную базу знаний.

17. Низкоуровневое программирование: система команд ЭВМ: регистры процессора, режимы адресации. Ассемблер.

Под **командой** понимают совокупность сведений, представленных в виде двоичных кодов, необходимых процессору для выполнения очередного шага. Множество реализуемых машинных действий образует систему команд. Система команд часто определяет области и эффективность применения ЭВМ. Состав и число команд должны быть ориентированы на стандартный набор операций, используемых пользователем для решения своих задач. По функциональному назначению в системе команд ЭВМ различают следующие группы:

Команды передачи данных.

Команды обработки данных (команды сложения, умножения, сдвига, сравнения).

Команды передачи управления (команды безусловного и условного перехода).

Команды дополнительные (типа RESET, TEST).

Группа команд передачи управления обеспечивает принудительное изменение порядка выполнения команд в программе.

Форматом команды называется заранее обговоренная структура полей в её кодах, позволяющая ЭВМ распознавать составные части кода. Главным элементом кода команды является код операции (КОП), что определяет, какие действия будут выполнены по данной команде. Команды процессора делятся на двухоперандные, однооперандные и безоперандные.

Регистры процессора: Регистр - это определенный участок сверхбыстрой памяти внутри самого процессора, который используется для промежуточного хранения информации, обрабатываемой процессором. Некоторые регистры содержат только определенную информацию.

Регистры общего назначения - EAX, EBX, ECX, EDX. Они 32-х битные и делятся еще на две части, нижние из которых AX, BX, CX, DX - 16-ти битные, и деляется еще на два 8-ми битных регистра. Так, AX делится на AH и AL, DX на DH и DL и т.д. Регистр CX чаще всего используется как счетчик в циклах.

Регистры сегментов - это CS, DS, ES, FS, GS, SS. CS - сегмент кода (страница памяти) исполняемой в данный момент программы. DS - сегмент (страница) данных исполняемой программы, т.е. константы, строковые ссылки и т.д. SS - сегмент стека исполняемой программы. ES, FS, GS - дополнительные сегменты, и могут не использоваться программой.

Регистры смещения - EIP, ESP, EBP, ESI, EDI. Эти регистры 32-х битные, нижняя половина которых доступна как регистры IP, SP, BP, SI, DI. EIP - указатель команд, и содержит величину смещения относительно начала программы на линию кода, которая будет исполняться следующей. То есть полный адрес на следующую исполняемую линию кода будет CS:EIP. Регистр ESP указывает на адрес вершины стека. Регистр EBP содержит адрес, начиная с которого в стек вносится или забирается информация (или "глубина" стека). Регистр ESI - адрес источника, и содержит адрес начала блока информации для операции "переместить блок" (полный адрес DS:SI), а регистр EDI- адрес назначения в этой операции (полный адрес ES:EDI).

Регистры управления памятью: GDTR – сод адрес и размер таблицы GDT. LDTR - сод адрес, размер LDT. IDTR - сод адрес и размер таблицы IDT. TR – сод адрес, размер TSS.

Управляющие регистры: CR0 Сод системные управляющие флаги. CR1 Зарезервирован. CR2 Сод линейный адрес, который вызвал ошибку обращения к странице памяти. CR3 (PDBR) Сод физический базовый адрес каталога страниц. CR4 Сод группу флагов, которые разрешают некоторые архитектурные расширения.

Регистр флагов FLAGS(EFLAGS) содержащий текущее состояние процессора. FLAGS - 16-битный регистр. Расширенные регистры EFLAGS и RFLAGS являются 32-битными и 64-битными соответственно.

Способы формирования адресов ячеек памяти можно разделить на **абсолютные и относительные. Абсолютные** способы формирования предполагают, что двоичный код адреса ячейки памяти может быть целиком извлечен либо из адресного поля команды, либо из какой-нибудь другой ячейки в случае косвенной адресации. **Относительные** способы формирования предполагают, что двоичный код адреса ячейки памяти образуется из нескольких составляющих: код базы, код индекса, код смещения.

Режим адресации памяти - это схема преобразования адресной информации об операнде в его исполнительный адрес.

Непосредственный. Непосредственная адресация предполагает, что операнд занимает одно из полей команды и, следовательно, выбирается из оперативной памяти одновременно с ней. **Регистровый**. Данное содержится в определяемом командой регистре. 16-битный операнд может находиться в регистрах AX, BX, CX, DX, SI, DI, SP или BP, а 8-битный - в регистрах AL, AH, BL, BH, CL, CH, DL или DH. **Прямой** предполагает, что эффективный адрес является частью команды.

При **косвенном режиме адресации** эффективный адрес операнда находится в базовом регистре BX, BP или одном из индексных регистров DI либо SI. При **относительном режиме адресации** эффективный адрес равен сумме содержимого базового или индексного регистра и 8- или 16-битного смещения. **Базово-индексный.** Эффективный адрес равен сумме содержимого базового и индексного регистров:

$$EA = \begin{Bmatrix}(BX)\\(BP)\end{Bmatrix} + \begin{Bmatrix}(SI)\\(DI)\end{Bmatrix}.$$

Относительный базово-индексный. Эффективный адрес равен сумме 8-или 16-битного смещения и базово-индексного адреса:

$$EA = \begin{Bmatrix}(BX)\\(BP)\end{Bmatrix} + \begin{Bmatrix}(SI)\\(DI)\end{Bmatrix} + \begin{Bmatrix}\text{8-битное смещение}\\\text{16-битное смещение}\end{Bmatrix}.$$

Язык ассемблера — тип языка программирования низкого уровня, представляющий собой формат записи машинных команд, удобный для восприятия человеком. Также язык ассемблера обеспечивает связывание частей программы и данных через метки, выполняемое при ассемблировании (для каждой метки высчитывается адрес, после чего каждое вхождение метки заменяется на этот адрес).

Поскольку системы команд микропроцессоров различаются, каждый процессор имеет свой набор команд на языке ассемблера и свои ассемблеры.

Достоинства языка ассемблера

- Максимально оптимальное использование средств процессора, использование меньшего количества команд и обращений в память, и как следствие — большая скорость и меньший размер программы
- Использование расширенных наборов инструкций процессора (MMX, SSE, SSE2, SSE3)
- Доступ к портам ввода-вывода и особым регистрам процессора (в большинстве ОС эта возможность доступна только на уровне модулей ядра и драйверов)
- Возможность использования самомодифицирующегося (в том числе перемещаемого) кода (под многими платформами эта возможность недоступна, так как запись в страницы кода запрещена, в том числе и аппаратно, однако в большинстве общедоступных систем из-за их врожденных недостатков имеется возможность исполнения кода содержащегося в сегменте (секции) данных, куда запись разрешена)
- Максимальная «подгонка» для нужной платформы

Недостатки

- Большие объемы кода, большое число дополнительных мелких задач, меньшее количество доступных для использования библиотек, по сравнению с языками высокого уровня
- Трудоёмкость чтения и поиска ошибок (хотя здесь многое зависит от комментариев и стиля программирования)
- Зачастую компилятор языка высокого уровня, благодаря современным алгоритмам оптимизации, даёт более эффективную программу.
- Непереносимость на другие платформы, кроме совместимых
- Ассемблер более сложен для совместных проектов

18. Операционные системы: архитектура, структура классического ядра ОС.

ОС - базовый комплект управляющих программ, которые вступают как интерфейс между аппаратурой компьютера, и предназначены для наиболее эффективного использования ресурсов вычислительной системы, организации надежных вычислений. Функции ОС: управление аппаратными средствами компьютера, работа с файлами, доступ к периферийным устройствам, управление оперативной и энергонезависимой памятью, взаимодействие между процессами, сетевое взаимодействие, выполнение прикладных программ и утилит, предоставление пользовательского интерфейса и т. п.

Совместимость: возможность использования ранее разработанного ПО в последующих разработках. Двоичная совместимость – возможность переноса бинарных файлов (Win). Совместимость исходного кода – возможность переноса исходного кода и перекомпиляции, система с заведомо открытым кодом (Unix). **Переносимость:** возможность запуска ОС на разном оборудовании. **Расширяемость:** возможность адаптации ОС к новым типам оборудования. **Надежность:** способность ОС работать без сбоев. **Отказоустойчивость:** сохранение работоспособности ОС после возникновения сбоя. **Безопасность:** возможность ограничения доступа пользователей или приложений по результатам выполнения определенных операций: разрешения, т.е. требуется специально запрещать, и запрет – требуется специально разрешать.

Архитектура. Ядро – поток, который получает события и передает их обработчику, работает в привилегированном режиме, защищено от приложений пользователя. Распределяет основные ресурсы системы, выполняет роль арбитра в споре приложений за ресурсы, аппаратура компьютера должна поддерживать как минимум два режима работы: режим пользователя, режим ядра.

Компоненты пользовательского режима имеют свои защищенные адресные пространства, потоки этих процессов выполняются в непривилегированном режиме процессора (называемом пользовательским), не могут выполнять привилегированные команды процессора, имеют ограниченный и опосредованный доступ к системным данным, и к системному адресному пространству, не имеют прямого доступа к оборудованию. Правда, в процессе своей работы, потоки этих процессов, вызывая системные сервисы, переходят в режим ядра, но в этом случае полностью теряют контроль над своим выполнением до возвращения обратно в режим пользователя. Процессы пользовательского режима рассматриваются как потенциально опасные с точки зрения стабильности системы.

Компоненты ядра разделяют единое адресное пространство, выполняются в привилегированном режиме процессора (называемом режимом ядра), могут выполнять все, в том числе и привилегированные, команды процессора, имеют неограниченный и прямой доступ к системным данным и коду, имеют прямой, или через HAL(hardware abstraction level), доступ к оборудованию. Все компоненты ОС работают в режиме ядра. Внутренний поток (ядро) является диспетчером событий.

Слои ядра: Аппаратная поддержка – прерывания устройств, механизмы переключения м/у уровнями (вентили, заглушки). **Аппаратно-зависимый слой** – компоненты ОС, реализованные на assembler абстрактные запросы устройствам, а тот слой реализует их в зависимости от железа. В идеале этот слой полностью изолирует вышележащие слои ядра от особенностей аппаратуры. Это позволяет разрабатывать вышележащие слои на основе машинно-независимых модулей. **Базовые механизмы** ядра. Этот слой выполняет наиболее примитивные операции ядра, например, диспетчеризацию прерываний, перемещение страниц памяти на диск. Модули данного слоя не принимают решений они только выполняют решения сверху, что позволяет называть их исполнительным механизмом или механизмами. **Исполнительный слой**. Здесь принимаются решения об активации той или иной задачи, предоставления доступа к ресурсам и т.д. **Менеджеры ввода/вывода**. Этот слой состоит их мощных функциональных модулей, организующих от абстракций до спец. устройств ввода-вывода. Обычно здесь представлены менеджеры процессов, ввода-вывода (в/в), файловой системы и оперативной памяти. Внутри слоя менеджеров существуют тесные взаимосвязи, отражающие факт, что для выполнения процессу нужен доступ одновременно к нескольким ресурсам. **Интерфейс системных вызовов** - «точка» входа в ядро. Этот слой является самым верхним слоем ядра и взаимодействует непосредственно с приложениями и системными утилитами, образуя прикладной программный интерфейс ОС. Функции API, обслуживающие системные вызовы, предоставляют доступ к ресурсам системы в удобной и компактной форме.

19. Функции ОС по управлению памятью, механизмы выделения памяти процессу, управление адресным пространством процесса.

Функциями ОС по управлению памятью в мультипрограммной системе являются:

- отслеживание свободной и занятой памяти;
- выделение памяти процессам и освобождение памяти по завершении процессов;
- вытеснение кодов и данных процессов из оперативной памяти на диск (полное или частичное), когда размеры основной памяти не достаточны для размещения в ней всех процессов, и возвращение их в оперативную память, когда в ней освобождается место;
- настройка адресов программы на конкретную область физической памяти.
- Динамическое выделение и освобождение памяти. Выделение памяти случайной длины в случайные моменты времени из общего пула памяти приводит к фрагментации, поэтому
- Дефрагментация памяти
- Защита памяти, состоит в том, чтобы не позволить выполняемому процессу записывать или читать данные из памяти, назначенной другому процессу.

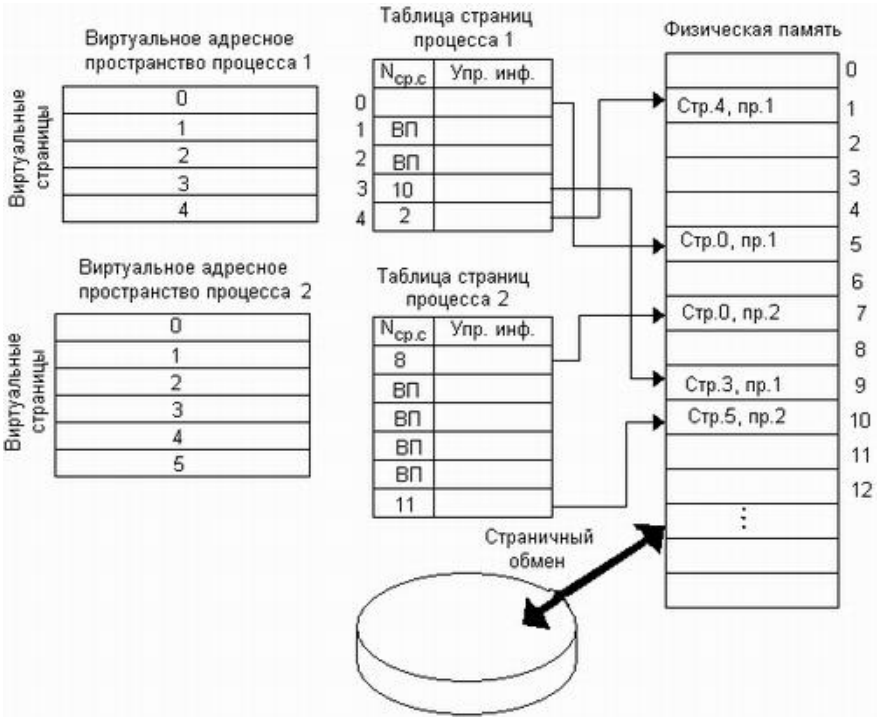
Механизмы выделения памяти процессу

все алгоритмы распределения памяти разделены на два класса: алгоритмы, в которых используется перемещение сегментов процессов между оперативной памятью и диском, и алгоритмы, в которых внешняя память не привлекается.

Виртуальное адресное пространство каждого процесса делится на части одинакового, фиксированного для данной системы размера, называемые виртуальными страницами (virtual pages). В общем случае размер виртуального адресного пространства процесса не кратен размеру страницы, поэтому последняя страница каждого процесса дополняется фиктивной областью.

Вся оперативная память машины также делится на части такого же размера, называемые физическими страницами (или блоками, или кадрами). Размер страницы выбирается равным степени двойки: 512, 1024, 4096 байт и т. д. Это позволяет упростить механизм преобразования адресов.

При создании процесса ОС загружает в оперативную память несколько его виртуальных страниц (начальные страницы кодового сегмента и сегмента данных). Копия всего виртуального адресного пространства процесса находится на диске.



Для каждого процесса операционная система создает таблицу страниц — информационную структуру, содержащую записи обо всех виртуальных страницах процесса.

Запись таблицы, называемая дескриптором страницы, включает следующую информацию:

номер физической страницы, в которую загружена данная виртуальная страница; * признак присутствия, устанавливаемый в единицу, если виртуальная страница находится в оперативной памяти; * признак модификации страницы, который устанавливается в единицу всякий раз, когда производится запись по адресу, относящемуся к данной странице; * признак обращения к странице, называемый также битом доступа, который устанавливается в единицу при каждом обращении по адресу, относящемуся к данной странице.

Сегментная организация. Разбиение виртуального адресного пространства на «осмысленные» части делает принципиально возможным совместное использование фрагментов программ разными процессами. Пусть, например, двум процессам требуется одна и та же подпрограмма, которая к тому же обладает свойством реентерабельности . Тогда коды этой подпрограммы могут быть оформлены в виде отдельного сегмента и включены в виртуальные адресные пространства обоих процессов. При отображении в

физическую память сегменты, содержащие коды подпрограммы из обоих виртуальных пространств, проецируются на одну и ту же область физической памяти. Таким образом оба процесса получат доступ к одной и той же копии подпрограммы.
Итак, виртуальное адресное пространство процесса делится на части — сегменты, размер которых определяется с учетом смыслового значения содержащейся в них информации. Отдельный сегмент может представлять собой подпрограмму, массив данных и т. п. Деление виртуального адресного пространства на сегменты осуществляется компилятором на основе указаний программиста или по умолчанию, в соответствии с принятыми в системе соглашениями. Максимальный размер сегмента определяется разрядностью виртуального адреса, например при 32-разрядной организации процессора он равен 4 Гбайт.
Сегментно-страничное распределение. Данный метод представляет собой комбинацию страничного и сегментного механизмов управления памятью и направлен на реализацию достоинств обоих подходов. Так же как и при сегментной организации памяти, виртуальное адресное пространство процесса разделено на сегменты. Это позволяет определять разные права доступа к разным частям кодов и данных программы.
Перемещение данных между памятью и диском осуществляется не сегментами, а страницами. Для этого каждый виртуальный сегмент и физическая память делятся на страницы равного размера, что позволяет более эффективно использовать память, сократив до минимума фрагментацию. В большинстве современных реализаций сегментно-страничной организации памяти все виртуальные сегменты образуют одно непрерывное линейное виртуальное адресное пространство.
2 способа задания адреса: 1-линейным виртуальным адресом, который равен сдвигу данного байта относительно границы общего линейного виртуального пространства, 2-парой чисел, одно из которых является номером сегмента, а другое — смещением относительно начала сегмента. обычно используют второй способ, так как он позволяет непосредственно определить принадлежность адреса некоторому сегменту и проверить права доступа процесса к нему.

20. Распределение процессорного времени в многозадачных системах. Потоки. Нити. Разделение ресурсов при мультипрограммировании.

Мультипрограммирование - одновременное выполнение нескольких задач на одном или нескольких процессорах. В мультипрограммировании ключевым местом является способ составления расписания, по которому осуществляется переключение между задачами (планирование), а также механизм, осуществляющий эти переключения.

По времени планирования можно выделить статическое и динамическое составление расписания. При **статическом планировании** расписание составляется заранее, до запуска приложений, и операционная система в дальнейшем просто выполняет составленное расписание. В случае **динамического планирования** порядок запуска задач и передачи управления задачам определяется непосредственно во время исполнения. Статическое расписание свойственно **системам реального времени**, когда необходимо гарантировать заданное время и сроки выполнения необходимых операций. В универсальных операционных системах статическое расписание практически не применяется. Выделяют динамическое планирование с использованием **квантов времени** - когда каждый выполняемой задаче назначают определенной продолжительности квант времени и планирование с использованием **приоритетов** - когда задачам назначают специфичные приоритеты и переключение задач осуществляют с учетом этих приоритетов.

Выделяют понятия **вытесняющей и невытесняющей** многозадачности: в случае невытесняющей многозадачности решение о переключении принимает выполняемая в данный момент задача, а в случае вытесняющей многозадачности такое решение принимается операционной системой (или иным арбитром), независимо от работы активной в данный момент задачи.

Процесс является объектом планирования адресного пространства и некоторых ресурсов, выделенных задаче. Но при этом процесс не является потребителем процессорного времени и не подлежит планированию проц. времени. **Поток(нить)** является объектом планирования процессорного времени. **Задача** - процесс и поток, выполняемый в нем.

Принято также деление потоков на потоки **ядра(тяжелые) и потоки пользователя(волокна)** Потоки ядра в данном контексте являются потоками, управляемыми планировщиком ядра операционной системы. Потоки пользователя управляются планировщиком пользовательского процесса. В Windows для обозначения этих понятий использованы термины process (процесс), thread (поток ядра) и fiber (волокно ползователя).

Объекты синхронизации. Критическая секция: это часть программы, в которой осуществляется доступ к разделяемым данным. По сути сигнальная переменная, находящаяся в 2-х состояних (свободен/занят). Если два потока полезут в критическую секцию, то ОС создаст дополнительный объект, но котором второй объект будет ожидать. Критические секции, подходят для синхронизации потоков одного процесса. Задачу синхронизации потоков различных процессов принято решать с помощью объектов ядра. Объекту ядра может быть присвоено имя, они позволяют задавать тайм-аут для времени ожидания и обладают еще рядом возможностей для реализации гибких сценариев синхронизации. Однако их использование связано с переходом в режим ядра.

Таймер.

События (автоматический/ручной сброс). Один выполняет, много ждут.

Семафоры: Семафоры обычно используются для учета ресурсов (текущее число задается переменной S) S не может быть больше максимального и отрицательным. Значение S, равное нулю, означает, что семафор занят.

Мьютекс: объект ядра, используются для взаимоисключающего доступа к одному ресурсу. Один поток может захватить мьютекс, остальные при этом ждут его освобождения. Если поток завершается, не освободив мьютекс, последний переходит в свободное состояние. Отличие от семафоров в том, что поток, занявший мьютекс, получает права на владение им. Только этот поток может освободить мьютекс. Поэтому мнение о мьютексе как о семафоре с максимальным значением 1 не вполне соответствует действительности.

21. Одноранговые сети и сети на основе серверов, домены. Протоколы и стандарты технологий локальных сетей.

Локальная вычислительная сеть (ЛВС) - это группа относительно небольшого количества компьютеров, объединенных совместно используемой средой передачи данных, расположенных на ограниченной по размерам небольшой площади.

В **одноранговой сети** все компьютеры равноправны: нет иерархии среди компьютеров и нет выделенного (dedicated) сервера. Как правило, каждый компьютер функционирует и как клиент, и как сервер;

Клиент - абонент сети, не отдающий своего ресурса в сеть, но имеющий доступ к ресурсам сети.

Сервер - абонент сети, отдающий в сеть свой ресурс, и имеющий или не имеющий доступа к ресурсам сети. Также сервером называют специализированный компьютер, предназначенный для работы в сети, предоставляющий клиентам специализированные ресурсы (быстродействующие диски большого объема, быстрый процессор, большую память).

\+ одноранговой сети: Простота, Невысокая стоимость, Поддержка встроена в ОС,

Одноранговая сеть вполне подходит там, где: количество пользователей не превышает 10 человек, пользователи расположены компактно, вопросы защиты данных не критичны, в обозримом будущем не ожидается значительного расширения сети.

Сеть на основе сервера - сеть, в которой имеется четкое разделение абонентов на клиентов и серверов, и в которой есть хотя бы один выделенный сервер. Выделенным называется такой сервер, который функционирует только как сервер (исключая функции клиента или рабочей станции) и не способный выполнять другие (не сетевые) задачи. Он специально оптимизирован для быстрой обработки запросов от сетевых клиентов и для управления защитой файлов и каталогов.

\+ сетей на сервере: Централизованное разделение ресурсов и администрирование, Защита данных, легче резервное копирование и репликация, масштабируемость

Домен (domain) - определенная администратором сети совокупность компьютеров, использующих общую базу данных и систему защиты; каждый домен имеет уникальное имя.

Модель OSI		
Тип данных	**Уровень**	**Функции**
http,ftp	7. Прикладной	Доступ к сетевым службам
Данные	6. Представления	Представление и кодирование данных
PPTP,SMPP	5. Сеансовый	Управление сеансом связи
Сегменты TCP,UDP	4. Транспортный	Прямая связь между конечными пунктами и надежность
Пакеты IP	3. Сетевой	Определение маршрута и логическая адресация
Кадры	2. Канальный	Физическая адресация
Биты	1. Физический	Работа со средой передачи, сигналами и двоичными данными

Ethernet,token ring

Ethernet, 802.11 WiFi

Иерархия протоколов

Для упрощения структуры большинство сетей организуются в наборы уровней или слоев, каждый последующий из которых возводится над предыдущим. Количество уровней, их названия, содержание и назначение разнятся от сети к сети.

Однако во всех сетях целью каждого уровня является предоставление неких сервисов для вышестоящих уровней. При этом от них скрываются детали реализации предоставляемого сервиса.

Правила и соглашения, используемые в данном общении, называются протоколом уровня п. По сути, протокол является договоренностью общающихся сторон о том, как должно происходить общение. Список протоколов, используемых системой, по одному протоколу на уровень, называется **стеком протоколов**.

Стандартизация сетей. IEEE (Institute of Electrical and Electronics Engineers). комитет IEEE 802.

Самые успешные рабочие группы - 802.3 (Ethernet) и 802.11 (Wireless networks)

22. Маршрутизация в сетях ЭВМ. Межсетевое взаимодействие на базе TCP/IP.

Задача **маршрутизации** - выбор маршрута для передачи от отправителя к получателю. Этот выбор производится в соответствии с реализуемым алгоритмом маршрутизации. Основные цели маршрутизации - минимальные задержки пакета при его передачи; максимальная пропускная способность сети; защита пакета; минимальная стоимость передачи пакета адресату.

Способы маршрутизации - **централизованная**, **децентрализованная**, **смешанная**. **Централизованная** маршрутизация реализуется обычно в сетях с централизованным управлением. Выбор маршрута для каждого пакета осуществляется в центре управления сетью, а узлы сети связи только воспринимают и реализуют результаты решения задачи маршрутизации. Такое управление маршрутизацией уязвимо к отказам центрального узла и не отличается высокой гибкостью.

Распределенная (**децентрализованная**) маршрутизация выполняется главным образом в сетях с децентрализованным управлением. Функции управления маршрутизацией распределены между узлами сети, которые располагают для этого соответствующими средствами. Распределенная маршрутизация сложнее централизованной, но отличается большей гибкостью.

Смешанная маршрутизация характеризуется тем, что в ней в определенном соотношении реализованы принципы централизованной и распределенной маршрутизации. К ней относится, например, гибридная адаптивная маршрутизация.

Данные о текущей топологии сети и пропускной способности линий связи предоставляются узлам без затруднений. Однако нет способа для точного предсказания состояния нагрузки в сети. Следовательно, во всех случаях алгоритмы маршрутизации выполняются в условиях неопределенности текущего и будущего состояний сети.

Различают 3 вида маршрутизации: **Простая** - при выборе маршрута не учитывается ни изменение топологии сети, ни изменение её состояния, нагрузки. Имеет низкую эффективность, преимущество - простота реализации алгоритма маршрутизации. Из этого вида практическое применение получили **случайная** и **лавинная** маршрутизации.

Фиксированная - при выборе маршрута учитывается изменение топологии сети и не учитывается изменение её нагрузки. Различают **однопутевую** и **многопутевую** фиксированные маршрутизации.

Адаптивная - принятие решения о направлении передачи пакетов осуществляется с учётом изменения как топологии, так и нагрузки сети. Есть **локальная**, **распределённая**, **централизованная**, и **гибридная** адаптивные маршрутизации.

Методы маршрутизации: **Выбор кратчайшего пути.** Строит граф подсети, узел - маршрутизатор, дуга — линия связи. Результат – кратчайший путь на графе.

Заливка Метод заливки представляет собой еще один статический алгоритм, при котором каждый приходящий пакет посылается на все исходящие линии, кроме той, по которой пришел пакет. **Маршрутизация по вектору расстояний** Алгоритмы маршрутизации по вектору расстояний работают, опираясь на таблицы (то есть векторы), поддерживаемые всеми маршрутизаторами и содержащие наилучшие известные пути к каждому из возможных адресатов. Для обновления данных этих таблиц производится обмен информацией с соседними маршрутизаторами.

Маршрутизация с учетом состояния линий В основе алгоритма лежит простая идея, ее можно изложить в пяти требованиях к маршрутизатору. Каждый маршрутизатор должен: 1. Обнаруживать своих соседей и узнавать их сетевые адреса. 2. Измерять задержку или стоимость связи с каждым из своих соседей. 3. Создавать пакет, содержащий всю собранную информацию. 4. Посылать этот пакет всем маршрутизаторам. 5. Вычислять кратчайший путь ко всем маршрутизаторам.

В результате каждому маршрутизатору высылаются полная топология и все измеренные значения задержек. После этого для обнаружения кратчайшего пути к каждому маршрутизатору может применяться алгоритм Дейкстры.

Иерархическая маршрутизация Размер таблиц маршрутов, поддерживаемых маршрутизаторами, увеличивается пропорционально увеличению размеров сети. Поэтому в больших сетях маршрутизация должна осуществляться иерархически, как это делается в телефонных сетях. При использовании иерархической маршрутизации маршрутизаторы разбиваются на отдельные так называемые регионы. Каждый маршрутизатор знает все детали выбора маршрутов в пределах своей области, но ему ничего не известно о внутреннем строении других регионов.

Широковещательная маршрутизация. Методы: прямая (нужны все адреса назначения), заливка, многоадресная маршрутизация (прямая с разбиением), на связующем дереве, продвижение по встречному пути.

Межсетевое взаимодействие на базе TCP/IP.

Протокол IP (Internet Protocol – межсетевой протокол). В каждой очередной сети , лежащей на пути перемещения пакета, протокол IP обращается к средствам транспортировки этой сети, чтобы с их помощью передать пакет на маршрутизатор, ведущий к следующей сети, или непосредственно на узел-получатель. Таким образом, одной из важнейших функций IP является поддержание интерфейса с нижележащими

технологиями сетей, образующих составную сеть. Кроме того, в функции протокола входит поддержание интерфейса с протоколами вышележащего транспортного уровня, в частности с протоколом TCP, который решает все вопросы обеспечения надежной доставки данных по составной сети в стеке TCP\IP.

Протокол IP относится к протоколам без установления соединений, он поддерживает обработку каждого IP-пакета как независимой единицы обмена, не связанной с другими IP-пакетами. В протоколе IP нет механизмов, обычно применяемых для обеспечения достоверности конечных данных. Если во время продвижения пакета происходит какая-либо ошибка, то протокол IP по своей инициативе ничего не предпринимает для исправления этой ошибки (протокол IP реализует политику доставки «по возможности»).

TCP — это транспортный механизм, предоставляющий поток данных, с предварительной установкой соединения, за счёт этого дающий уверенность в достоверности получаемых данных, осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий одного пакета. В отличие от UDP гарантирует, что приложение получит данные точно в такой же последовательности, в какой они были отправлены, и без потерь.

23. Глобальные сети. Глобальные сети на выделенных линиях. Глобальные сети с коммутацией каналов. Глобальные сети с коммутацией пакетов.

Глобальная сеть (wide area network, WAN) охватывает значительную географическую область, часто целую страну или даже континент. Хосты соединяются коммуникационными подсетями. Хосты обычно являются собственностью клиентов (то есть просто клиентскими компьютерами), в то время как коммуникационной подсетью чаще всего владеет и управляет телефонная компания или поставщик услуг Интернета.

В большинстве глобальных сетей подсеть состоит из двух раздельных компонентов: линий связи и переключающих элементов. Линии связи, также называемые каналами или магистралями, переносят данные от машины к машине. Переключающие элементы – маршрутизаторы.

Передача по выделенным каналам – когда для связей абонентов желательно иметь собственные каналы связи. Если компьютеры находятся близко друг к другу, то обычно используется моноканал. На большие расстояния приходится использовать существующие линии связи. Такая связь стоит очень дорого, она очень надёжная, имеет высокое качество связи. Недостаток - КПД 1-3 процента.

Коммутация каналов (КК, *circuit switching*) — организация составного канала через несколько транзитных узлов из нескольких последовательно «соединённых» каналов на время передачи сообщения (*оперативная коммутация*) или на более длительный срок (*постоянная/долговременная коммутация* — время коммутации определяется административно, то есть пришёл техник и скоммутировал канал).

Условием того, что несколько физических каналов при последовательном соединении образуют единый физический канал, является равенство скоростей передачи данных в каждом из составляющих физических каналов. Равенство скоростей означает, что коммутаторы такой сети **не должны буферизовать передаваемые данные**. В сети с коммутацией каналов перед передачей данных всегда необходимо выполнить **процедуру установления соединения**, в процессе которой и создается составной канал. И только после этого можно начинать передавать данные.

+: Постоянная и известная скорость передачи, Низкий и постоянный уровень

-: Отказ сети в обслуживании запроса на установление соединения («Занято»), Нерациональное использование пропускной способности физических каналов, Обязательная задержка перед передачей данных из-за фазы установления соединения.

Коммутация сообщений (КС, *message switching*) — разбиение инфы на сообщения, кот. передаются последовательно к ближайшему транзитному узлу, который приняв сообщение, запоминает его и передаёт далее. Минус – большой размер буфера роутера (должен вместить сообщение).

Коммутация пакетов (КП, *packet switching*) — разбиение сообщения на «пакеты», которые передаются отдельно. Разница между сообщением и пакетом: размер пакета ограничен технически, сообщения — логически. При этом, если маршрут движения пакетов между узлами определён заранее, говорят о *виртуальном канале* (с установлением соединения, напр. IP). Если же для каждого пакета задача нахождения пути решается заново, говорят о *датаграммном* (без установления соединения) способе пакетной коммутации.

Коммутаторы пакетной сети отличаются от коммутаторов каналов тем, что они **имеют внутреннюю буферную память** для временного хранения пакетов, если выходной порт коммутатора в момент принятия пакета занят передачей другого пакета.

Сеть с коммутацией пакетов замедляет процесс взаимодействия конкретной пары абонентов, но повышает пропускную способность сети в целом.

+: Высокая общая пропускная способность сети при передаче пульсирующего трафика, Возможность динамически перераспределять пропускную способность физических каналов.

-: Неопределенность скорости передачи данных между абонентами, Переменная величина задержки пакетов, Возможные потери данных из-за переполнения буферов.

24. Сетевые операционные системы, функциональные компоненты. Сетевые службы и сервисы. Типичные сетевые службы и используемые протоколы.

Два значения термина «Сетевая ОС»: Во-первых, как совокупность ОС всех компьютеров сети.
Во-вторых, как операционная система отдельного компьютера, способного работать в сети.

Функциональные компоненты сетевой ОС:

- *Средства управления локальными ресурсами* компьютера реализуют все функции ОС автономного компьютера (распределение оперативной памяти между процессами, планирование и диспетчеризацию процессов, управление процессорами в мультипроцессорных машинах, управление внешней памятью, интерфейс с пользователем и др.);
- *Сетевые средства*, в свою очередь можно разделить на три компонента: 1) Средства предоставления локальных ресурсов и услуг в общее пользование – *серверная часть ОС;* 2) Средства запроса доступа к удаленным ресурсам и услугам- *клиентская чать ОС*; 3) *Транспортные средства ОС,* которые совместно с коммуникационной системой обеспечивают передачу сообщений между компьютерами сети.

Очень удобной и полезной функцией клиентской части ОС является способность отличить запрос к удаленному файлу от запроса к локальному файлу. Если клиентская часть умеет делать это, то приложения не должны заботиться о том, с локальным или удаленным файлом они работают, - клиентская программа сама распознает и перенаправляет запрос к удаленной машине.

Клиентские части сетевых ОС выполняют также преобразование форматов запросов к ресурсам. Они принимают запросы от приложений на доступ к сетевым ресурсам в локальной форме, то есть в форме, принятой в локальной части ОС. В сеть же запрос передается клиентской частью в другой форме, соответствующей требованиям серверной части ОС, работающей на компьютере, где расположен требуемы ресурс. Клиентская часть также осуществляет прием ответов от серверной части и преобразование их в локальный формат, так что для приложения выполнение локальных и удаленных запросов неразличимо.

Сетевые службы и сервисы. Совокупность серверной и клиентской частей ОС, предоставляющих доступ к конкретному типу ресурса компьютера через сеть, называется *сетевой службой*.

Говорят, что сетевая служба предоставляет пользователям сети некоторый набор *услуг*. Эти услуги иногда называют также *сетевым сервисом*. Далее под «службой» понимается сетевой компонент, который реализует некоторый набор услуг, а под «сервисом» - описание того набора услуг, который предоставляется данной службой. Таким образом, сервис – это интерфейс между потребителем услуг и поставщиком (службой).

Среди сетевых служб можно выделить такие, которые ориентированны не на простого пользователя, а на администратора. Такие службы используются для организации работы сети.

Любая сетевая служба содержит в своем составе две несимметричные части - клиентскую и серверную. Сетевая служба может быть представлена либо обеими частями, либо только одной.

Типичные сетевые службы (на примере серверной ОС Windows 2000):

Сервер (Server) – позволяет системе предоставлять в совместное пользование ее ресурсы, например, файлы и принтеры; **Рабочая станция (Workstantion)** – предоставляет системе доступ к общим ресурсам другого компьютера; **Обозреватель компьютеров (Computer Browser)** – поддерживает список общих ресурсов сети; **Служба сообщений (Messenger)** – позволяет системе выводить выплывающие сообщения о событиях в других системах сети; **Оповещатель (Alerter)** – совместно со службой сообщений рассылает избранным пользователям административные оповещения; **Netlogon** – обеспечивает защищенные каналы связи между компьютерами Windows для обмена информацией, связанной с процессом авторизации; **IIS** – обеспечивает работу служб Интернета, например, Web- и FTP-сервера; **WINS** – преобразует NetBios-имена в IP-адреса; **DNS-Server** – преобразует DNS-имена компьютера в IP-адреса; **DHCP-Server** – автоматически настраивает параметры TCP\IP в сетях с многочисленными клиентскими системами; **RRA Service** – позволяет направлять трафик между двумя ЛВС или ГВС и ЛВС; обеспечивает поддержку различных протоколов маршрутизации; **Распределенная файловая система DFS (Distributed File System**) – позволяет представлять все общие диски на серверах сети в виде единого общего ресурса. **Microsoft Cluster Server** – позволяет системам под управлением Windows NT 4.0 Enterprise Server или Windows 2000 Advanced Server функционировать в составе кластера – группы компьютеров, одновременно выполняющих одну и ту же программу, например, для повышения отказоустойчивости.

25. Сетевые файловые системы, файловые серверы с сохранением состояния и без, кэширование, репликация. Служба каталогов, Межсетевое взаимодействие.

Сетевая файловая система (ФС) в общем случае включает следующие элементы:
Локальная файловая система, Интерфейс локальной файловой системы, Сервер сетевой файловой системы, Клиент сетевой файловой системы, Интерфейс сетевой файловой системы, Протокол клиент-сервер сетевой файловой системы.
Клиенты сетевой ФС – это программы, которые работают на многочисленных компьютерах, подключенных к сети. В качестве таких приложений часто выступают графические или символьные оболочки ОС, а также любые пользовательские программы.
Файловый сервер может быть реализован по одной из двух схем: с запоминанием данных о файловых операциях клиента, то есть по схеме **stateful**, и без запоминания таких данных, то есть по схеме **stateless**.
Севреры stateful работаю по схеме, обычной для любой локальной файловой службы. Такой сервер поддерживает от же набор вызовов, что и локальная система, то есть вызовы open, read, write, seek и close. Открывая файлы по вызову open, переданному по сети клиентом, сервер stateful должен запоминать какие файлы открыл каждый пользователь в своей внутренней таблице. Обычно при открытии файла клиентскому приложению по сети возвращается дескриптор файла или другое число, которое используется при последующих вызовах для идентификации файла. При поступлении вызова read, write или seek сервер использует дескриптор файла для определения, какой файл нужен. В этой таблице хранится также значение указателя на текущую позицию в файле, относительно которой выполняется операция чтения или записи. Таблица, отображающая дескрипторы файлов на сами файлы, является хранилищем информации о состоянии клиентов.
Для сервера stateless каждый запрос должен содержать исчерпывающую информацию (полное имя файла, смещение в файле и т.п.), необходимую серверу для выполнения требуемой операции. Эта информация увеличивает длину сообщения и время, которое тратит сервер на локальное открытие файла, каждый раз, когда над ним производится очередная операция чтения или записи. Серверы, работающие по схеме stateless, не поддерживают в протоколе с клиентами таких операций как открытие и закрытие файлов.
+ stateless: Отказоустоичивы, Не нужны вызовы open\close, Меньше памяти сервера расходуется на таблицы, Нет ограничения на число открытых файлов, Отказ клиента не создает проблем для сервера
+ stateful: Более короткие сообщения при запросах, Лучшая производительность, Возможно опережающее чтение, Возможна блокировка файлов
Кэширование данных в оперативной памяти может существенно повысить скорость доступа к файлам, хранящимся на дисках. Кэширование также широко используется в сетевых файловых системах, где оно позволяет не только повысить скорость доступа с удаленным данным, но и улучшить масштабируемость и надежность файловой системы. Схемы кэширования, применяемые в СФС, отличаются по трем ключевым вопросам: 1) Месту расположения КЭШа (память сервера, диск клиента, память клиента); 2) Способу распространения модификаций (алгоритм сквозной записи, алгоритм отложенной записи, алгоритм «запись по закрытию»); 3) Проверке достоверности КЭШа (инициирование проверки клиентом, инициирование проверки сервером);
Репликация. Репликация подразумевает существование нескольких копий одного и того же файла, каждая из которых хранится на отдельном файловом сервере, при этом обеспечивается автоматическое согласование данных в копиях файлов. Иногда репликацией занимается отдельная служба ОС. Причины исп-я репликации: 1) Увеличение надежности за счет наличия независимых копий каждого файла, хранящихся на разных файловых серверах. При отказе одного из них файл остается доступным. 2) Распределение нагрузки между несколькими серверами.
Служба каталогов (справочная служба) - хранит информацию обо всех пользователях и ресурсах сети в виде унифицированных объектов с определенными атрибутами, а также позволяет отражать взаимосвязи между хранимыми объектами, такие как принадлежность пользователей к определенной группе, права доступа пользователей к компьютерам, вхождение нескольких узлов в одну подсеть, коммуникационные связи между подсетями, производственную принадлежность серверов и т.д.. Служба каталогов позволяет выполнять над хранимыми объектами набор некоторых базовых операций, таких как добавление и удаление объекта, включение объекта в другой объект, изменение значений атрибутов объекта, чтение атрибутов и некоторые другие. Обычно над службой каталогов строятся различные специфические сетевые приложения, которые используют информацию службы для решения конкретных задач: управления сетью, аутентификации пользователей, обеспечения прозрачности служб и других, перечисленных выше. Служба каталогов обычно строится на основе модели клиент-сервер: серверы хранят базу справочной информации, которой пользуются клиенты, передавая серверам по сети соответствующие запросы. Для клиента службы каталогов она представляется единой централизованной системой, хотя большинство хороших служб каталогов имеют распределенную структуру, включающую большое количество серверов, но эта структура для клиентов прозрачна.
Межсетевое взаимодействие. Нормой сегодняшнего дня являются неоднородные (гетерогенные) сети, состоящие из разнотипных рабочих станций, операционных систем и приложений, в которых для реализации

взаимодействия между компьютерами используются различные концентраторы, коммутаторы и маршрутизаторы. Отсюда следует очень важное требование, предъявляемое к современным сетевым ОС – способность интеграция с другими ОС.
В контексте межсетевого взаимодействия под термином «сеть» понимается *совокупность компьютеров, общающихся друг с другом с помощью единого стека протоколов.*
Самыми общими подходами к согласованию протоколов являются:

- Трансляция
- Мультиплексирование
- Инкапсуляция (туннелирование)

Трансляция обеспечивает согласование стеков протоколов путем преобразования сообщений, поступающих от одной сети в формат сообщений другой сети. Транслирующий элемент, в качестве которого могут выступать, например, программный или аппаратный шлюз, мост, коммутатор или маршрутизатор, размещается между взаимодействующими сетями и служит посредником в их «диалоге».
+:шлюзы сохраняют в неизменном виде программное обеспечение на клиентских компьютерах.
-: как централизованный ресурс, шлюз снижает надежность сети. Ухудш. масштабируемость.
Мультиплексирование стека протоколов заключается в том, что в сетевое оборудование или операционные системы серверов и рабочих станций встраиваются несколько стеков протоколов. Это позволяет клиентам и серверам выбирать для взаимодействия тот протокол, который является для них общим.
Инкапсуляция протоколов (туннелирование) - это еще один метод решения задачи согласования сетей, который, однако, применим только для согласования транспортных протоколов и только при определенных ограничениях. Инкапсуляция может быть использована, когда две сети с одной транспортной технологией необходимо соединить через транзитную сеть с другой транспортной технологией.
В процессе инкапсуляции принимают участие три типа протоколов: 1) Протокол-«пассажир» 2) Несущий протокол 3) Протокол инкапсуляции.

Транспортный протокол объединяемых сетей является протоколом-пассажиром, а протокол транзитной сети – несущим протоколом. Пакеты протокола-пассажира помещаются в поле данных пакетов несущего протокола с помощью протокола инкапсуляции. Пакеты протокола-пассажира никаким образом не обрабатываются при транспортировке их по транзитной сети. Инкапсуляцию выполняет пограничное устройство (обычно маршрутизатор или шлюз), которое располагается на границе между исходной и транзитной сетями. Извлечение пакетов-пассажиров из несущих пакетов выполняет второе пограничное устройство, которое находится между транзитной сетью и сетью назначения. Пограничные устройства указывают в пакетах свои адреса, а не адреса узлов назначения.
В связи с большой популярностью Интернета и стека TCP\IP несущим протоколом транзитной сети все чаше выступает протокол IP, а в качестве протоколов-пассажиров – все остальные протоколы локальных сетей (как маршрутизируемые так и не маршрутизируемые).

26. Высокопроизводительные коммуникационные среды. Обзор распространенных коммуникационных сред.

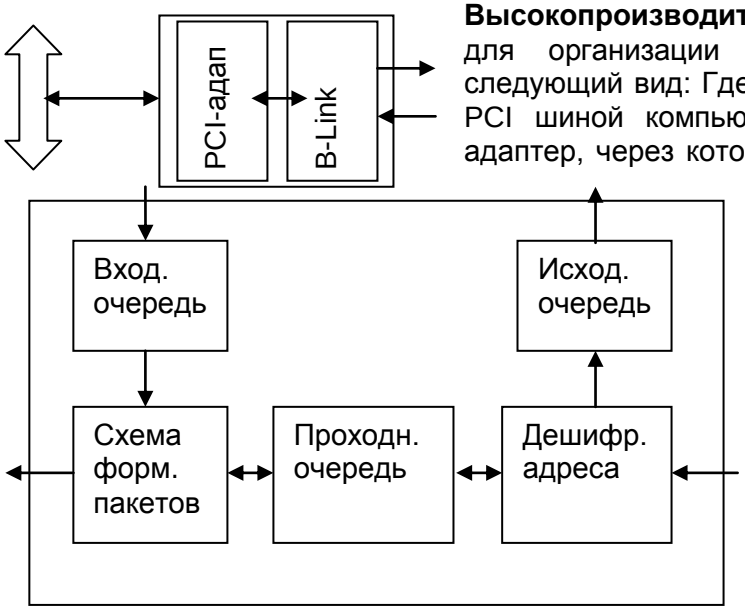
Применяются при решении задач, требующих большого количества пересылок коротких сообщений между узлами, в таких задачах время задержки (латентность) играет решающую роль.

Высокопроизводительная коммуникационная среда SRC (коннектор шин PCI). Идея реализации SRC в том, чтобы подключить удаленные компьютеры так, чтобы работать с ними (в частности, с памятью), можно было бы как с локальным устройством (устройство, подключенное через PCI шину).

Реализация оказалась таковой: при подключении через это устройство несколько компьютеров(у каждого – своя “карточка”), каждый из включенных компьютеров видит остальные, как некое дополнительное устройство на его PCI карте. Это происходит с каждым компьютером.

По сетевой топологии – это двунаправленное кольцо. Сигнал с адаптеров попадает во входные очереди, информация из которых попадает во входной буфер. Из буфера информация попадает в интерфейс карты PCI и информация по PCI шине передается процессору/памяти. Если же информация предназначается другому компьютеру – то она попадает в кэш незавершенных транзакций, откуда через устройство выбора линка попадает на соответствующий интерфейс линка и уходит на следующий интерфейс (соединенный компьютер).

Проблемы: каждое устройство на шине требует свое адресное пространство, а в данной реализации, каждый компьютер должен выделить много памяти, для реализации Shared Memory, памяти может не хватить для самого компьютера. Другая проблема – в определении адреса назначения, т.к. в шине каждого компьютера один и тот же будет под разным адресом.



Высокопроизводительная коммуникационная среда SCI. Например, для организации кольца(одностороннего) аппаратура будет иметь следующий вид: Где PCI адаптер сопряжения служит соединением между PCI шиной компьютера и рассматриваемым устройством, а B-Link – адаптер, через который можно соединять компьютеры. B-Link адаптер по структуре выглядит так

Сперва дешифруется адрес. Если адрес адресата – текущий компьютер, то идет передача в исходящую очередь – и передача по PCI шине, если же нет, то отправляется в проходящую очередь, после чего после формирования нового пакета – передается дальше. Данные, которые необходимо передать попадают в схему формирования пакетов через входную очередь и отправляются дальше.

Выше рассмотрена для организации кольца. Также существуют реализации, для организации двумерного тора (посредством увеличения количество B-Link-ов).

Высокопроизводительная коммуникационная среда GigaRing. Была реализована (по сетевой типологии) – как два встречных кольца. Был реализован механизм анализа входящего сигнала (грубо говоря, ключ), который будет пускать информацию через менее загруженный канал. Есть две очереди (для каждого направления по одному), а также по 2-а буфера(опять же для кажд. напр.) ввода и вывода информации с PCI шины.

В пределе это свелось к реализации SCI, поэтому технология не стала развиваться дальше.

Высокопроизводительная коммуникационная среда Myrinet. Поток символов – не зависит от размеров передаваемой информации – поток рассматривается как последовательность байт (9 бит каждый, 1ый – управляющий. Если =0, то это управляющая команда, если !=0, то это данные).

Передача – это двунаправленный поток информации, т.к. корректность определяется ответом. Имеет коммутируемую топологию, элементом коммутации является матрица 8x8. Коммутаторы на её основе поддерживают до 128 портов.

Формируется пакет таким образом, что записано каким путем идти данным.(Порт1, Порт3, Порт6). И пакет посылается от коммутатора коммутатору (указанные в “маршрутном листе”). Очередной коммутатор, прочитав информацию о том,куда надо передавать информацию, удаляет соответствующий сегмент из присланного потока данных и отправляет на соответствующий порт. В этой реализации именно отправитель думает о том, как быстрее и лучше передать сигнал – коммутаторы только анализируют ЕГО данные и передают на свои порты – не более.

В PCI-адаптере находится приливно-отливной буфер. Он предназначен для регулирования потока данных. Если данных очень много и аппаратура не справляется – передаем сигнал STOP, разбираемся с потоком, а потом возобновляем предачу последством отсылки команды GO.

27. Виды параллелизма. Пиковая и реальные производительности. Эффективность. Законы Амдаля. Классификация многопроцессрных ЭВМ.

В настоящее время существуют два основных подхода к распараллеливанию вычислений. Это параллелизм данных и параллелизм задач. Проблемы: равномерная загрузка процов, скорость обмена инфой м-ду процами. Уровни параллелизма: **Параллелизм заданий** — каждый процессор загружается своей собственной независимой от других вычислительной задачей. **Параллелизм на уровне программы** — вычислительная программа разбивается на части, которые могут выполняться одновременно на различных процессорах; **Параллелизм команд** — обычно реализован на низком уровне, это, например, конвейеры и т.д.; **Параллелизм на уровне машинных слов и арифметических операций** — например, логическое сложение выполняется одновременным сложением всех двоичных разрядов.

Архитектуры процов: 1) Классические (фон-нейман) 2) Суперскалярный процессор может выполнять несколько операций за один такт, снабжен логикой, позволяющей определить, являются ли команды независимыми, и достаточное число исполняющих устройств. В исполняющих устройствах могут быть конвейеры. Суперскалярные процессоры реализуют параллелизм на уровне команд. 3)Векторный процессор "умеет" обрабатывать одной командой не одно единственное значение, а сразу массив (вектор) значений.

Для вычисления производительности вводят следующие обозначения: T — время выполнения. N — число операций. t — время выполнения отдельной операции. s — число процев. r — производительность. pi — пиковая производительность, al — весовой коэффициент.

$T = N*t$. $r = N/T = 1/t$ $al_i = pi_i / (pi_1 + pi_{2+...})$; $al_1 + al_2 \ldots = 1$

R = T1/Tn, R — ускорение работы, T1/Tn – время работы на 1 и на n процессорах.

Законы Амдаля:

1) В общем случае производительность системы определяется наиболее медленным устройством.
2) Если B — доля последовательных вычислений, то $R_{max} = s / (B*s + 1 - B)$

В любом случае максимально возможное ускорение не превышает 1 / B.

Классификация многопроцессрных ЭВМ: Последовательные – классические компьютеры

SMP(симметричный мультипроцессор): процессоры независимы, каждый может обращаться к любой области памяти. При количестве процессоров > 16 слишком большие проблемы с когерентностью кэшей.

MPP (Массивно-параллельные компьютеры): процессор имеет свою память и не может обращаться к чужой памяти. Есть возможность передачи информации от одного процессора к другому. Нет проблем с кэшами.

NUMA/ccNUMA (неоднородная память): каждый процессор имеет свою память, но может обратиться и к чужой. (чуть медленнее), ccNUMA – аппаратно решает проблемы кэшей. В NUMA это делает программист.

Кластеры – отдельные компьютеры (часто – SMP), у каждого своя ОС, как-то связаны.

Классификация по Флинну:

SIMD – single-instruction, multiple-data stream processing (SuperScalar)

MIMD - multiple instruction, multiple data stream processing (кластр, SMP машина)

SISD - single-instruction, single data stream processing (обычные PC)

MISD - multiple instruction, single data stream processing (не существует)

28. Многопроцессорные ЭВМ и многомашинные комплексы. Обзор разных типов доступа к памяти. Общая память, распределенная память, MPP, кластеры.

Классификация многопроцессрных ЭВМ: Последовательные – классические компьютеры
SMP(симметричный мультипроцессор): процессоры независимы, каждый может обращаться к любой области памяти. При количестве процессоров > 16 слишком большие проблемы с когерентностью кэшей.
MPP (Массивно-параллельные компьютеры): процессор имеет свою память и не может обращаться к чужой памяти. Есть возможность передачи информации от одного процессора к другому. Нет проблем с кэшами.
NUMA/ccNUMA (неоднородная память): каждый процессор имеет свою память, но может обратиться и к чужой. (чуть медленнее), ccNUMA – аппаратно решает проблемы кэшей. В NUMA это делает программист.
Кластеры – отдельные компьютеры (часто – SMP), у каждого своя ОС, как-то связаны.
Классификация по Флинну: 1) SIMD – single-instruction, multiple-data stream processing (SuperScalar) 2) MIMD - multiple instruction, multiple data stream processing (кластр, SMP машина) 3) SISD - single-instruction, single data stream processing (обычные PC) 4) MISD - multiple instruction, single data stream processing (не существует)
Две принципиальные модели работы с памятью: 1. системы с разделяемой памятью (общая память) 2. системы с распределенной памятью (модель обмена сообщений), у которых каждый процессор имеет свою локальную оперативную память и к этой памяти у других процессоров нет доступа. Необходимо использовать какие-то механизмы для передачи данных от одних процессоров к другим.
Модель общей памяти (Shared Memory): Память – общая для всех процессоров. Данные – глобальны. Нужны дополнительные средства синхронизации: семафоры и критические секции. Простейший способ создать многопроцессорный вычислительный комплекс с разделяемой памятью - взять несколько процессоров, соединить их с общей шиной и соединить эту шину с оперативной памятью. Несколько улучшить картину может применение кэш-памяти для хранения команд. В результате этого уменьшается количество обращений к шине и быстродействие системы возрастает. Но возникает новая проблема - проблема *кэш-когерентности*.
Модель обмена сообщений. В этом случае отпадает необходимость в шине или переключателе. Нет и конфликтов по доступу к памяти, так как каждый процессор работает только со своей собственной памятью. Нет присущих системам с разделяемой памятью ограничений на число процессоров, нет, разумеется, и проблемы с кэшкогерентностью. Но, с другой стороны, возникают проблемы с организацией обмена данными между процессорами. Обычно такой обмен осуществляется при помощи обмена сообщениями посылками, содержащими данные. Для формирования такой посылки требуется время, для получения и считывания полученных данных тоже требуется определенное время. Эти дополнительные затраты времени - плата за все те преимущества, о которых шла речь.
Известны три системы, предоставляющие программисту удобный интерфейс для написания программ для систем с **распределённой памятью**.
Linda — создаётся видимость наличия общеё ассоциативной памяти, при помощи которой осуществляется взаимодействие и распараллеливание. Система имеет высокую степень абстракции и низкую производительность.
MPI (Message Passing Interface) — позволяет приложениям, запущенным на разных компьютерах, обмениваться сообщениями. Предоставляется большой набор функций. Низкая степень абстракции, высокая производительность.
PVM (Parallel Virtual Machine) — система предоставляет виртуальную параллельную машину, взаимодействие между процессами которой осуществляется при помощи сообщений и сигналов (как в UNIX). В PVM для управления процессами используется интерфейс UNIX, число процессов не связано с числом процессоров. Т.о., PVM даёт более высокий уровень абстракции, нежели MPI, однако производительность при этом остаётся достаточно высокой.
Компьютеры с SMP и NUMA архитектурами позволяют процессам взаимодействовать при помощи общей памяти. Для программиста это очень удобно, однако подобные архитектуры сильно ограничивают число процессоров, которые могут быть эффективно использованы в одной машине. Поэтому большие параллельные системы обычно основаны на архитектурах MPP или кластерах. Для взаимодействия между узлами используются сообщения, передаваемые через локальную сеть или какое-то специфическое коммуникационное оборудование, например, SCI.

29. Параллельная реализация прямых и итерационных методов решения линейных систем: метод Гаусса, LU-разложение для симметричных матриц.

Метод Гаусса может быть применен к системам линейных уравнений с произвольным числом уравнений и неизвестных. МГ основывается на возможности выполнения эквивалентных преобразований *линейных уравнений*(не меняют решение слау).

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n = b_m \end{cases}$$

МГ - это последовательное выполнение двух этапов. **Прямой ход:** На итерации i (0≤i<m-1) метода производится исключение неизвестной i для всех уравнений с номерами k > i (т.е. i<k≤m-1). Коэффициенты при неизвестных и свободные члены в последних m-1 уравнениях системы, определяются формулами:

$$a_{ij}^{(1)} = a_{ij} - \frac{a_{i1}}{a_{11}} \cdot a_{1j}; \qquad b_i^{(1)} = b_i - \frac{a_{i1}}{a_{11}} \cdot b_1 \quad (i, j = 2 \ldots m)$$

$a_{ij}^{(2)} = a_{ij}^{(1)} - \frac{ai2^{(1)}}{a22} * a_{2j}^{(1)}$, $b_i^{(2)} = b_i^{(1)} - \frac{ai2^{(1)}}{a22} * b_2^{(1)}$ (i, j = 3…m)

…

$a_{ij}^{(m-1)} = a_{ij}^{(m-2)} - \frac{ai,m-1^{(m-2)}}{am-1,m-1^{(m-2)}} * a_{m-2,j}^{(m-2)}$, $b_i^{(m-1)} = b_i^{(m-2)} - \frac{ai,m-1^{(m-2)}}{am-1,m-1^{(m-2)}} * b_{m-1}^{(m-2)}$ (i = j = m)

После проведения (m-1) подобной итерации матрица становится приведенной к верхнему треугольному виду. Строка, которая используется для исключения неизвестных наз *ведущей*, а диагональный элемент ведущей строки наз *ведущим элементом*. Выполнение вычислений является возможным только, если ведущий элемент имеет ненулевое значение. Если ведущий элемент $a_{i,i}$ имеет малое значение, то накапливается вычислительная погрешность .

Возможный способ избежать проблемы: при выполнении каждой очередной итерации следует определить коэффициент с максимальным значением по абсолютной величине в столбце, соответствующем исключаемой неизвестной и выбрать в качестве ведущей строку, в которой этот коэффициент располагается (метод главных элементов). Вычислительная сложность прямого хода *алгоритма Гаусса* с выбором ведущей строки имеет порядок $O(n^3)$.

Обратный ход: решение треугольной системы. Из последнего уравнения находим x_m. По найденному x_m из (m-1) уравнения находим x_{m-1}. Затем по x_{m-1} и x_m из (m-2) уравнения находим x_{m-2}. Процесс продолжаем, пока не найдем x_1 из первого уравнения. Вычислительная сложность обратного хода $O(n^2)$.

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1m}x_m = b_1 \\ a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 + \ldots + a_{2m}^{(1)}x_m = b_2^{(1)} \\ \ldots \\ a_{kk}^{(k-1)}x_k + a_{km}^{(k-1)}x_m = b_k^{(k-1)} \end{cases}$$

Если у нас число уравнений меньше числа неизвестных, то мы придем не к треугольной системе, а к ступенчатой.

так как прямой ход метода Гаусса прервется, когда уравнения закончатся, а неизвестные еще останутся. В таком случае в каждом уравнении системы перенесем все члены с неизвестными $x_{k+1},\ldots,x_m$ в правую часть.

Придавая неизвестным $x_{k+1},\ldots,x_m$ (называемым свободными) произвольные значения, получим треугольную систему из которой последовательно найдем все остальные неизвестные (наз базисными). Так как произвольные значения можно придавать любыми способами, система будет иметь бесчисленное множество значений.

LU-разложение: Матрица является симметричной, если она совпадает со своей транспонированной матрицей . Матрица представляется в виде произведения нижнетреугольной и верхнетреугольной A = LU (в общем случае PLU, где P – матрица перестановок (где каждый столбец и каждая строка содержит только один единичный элемент, все остальные = 0). *LU* – разложение может быть построено с использованием описанного выше метода Гаусса. Рассмотрим k - ый шаг метода Гаусса, на котором осуществляется обнуление поддиагональных элементов k - го столбца матрицы $A^{(k-1)}$. Как было описано выше, с этой целью используется следующая операция:

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - \mu_i^{(k)} a_{kj}^{(k-1)}, \qquad \mu_i^{(k)} = \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}, \quad i = \overline{k+1, n}, \quad j = \overline{k, n}.$$

В терминах матричных операций такая операция эквивалентна умножению $A^{(k)} = M_k A^{(k-1)}$, где элементы матрицы M_k определяются следующим образом

$m_{ij}^k = \begin{cases} 1, & i = j \\ 0, & i \neq j, \quad j \neq k \\ -\mu_{k+1}^{(k)}, & i \neq j, \quad j = k \end{cases}$. Т.е. матрица M_k имеет вид $\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -\mu_{k+1}^{(k)} & 1 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & -\mu_n^{(k)} & 0 & 0 & 1 \end{pmatrix}$.

При этом выражение для обратной операции запишется в виде $A^{(k-1)} = M_k^{-1} A^{(k)}$, где

$M_k^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & \mu_{k+1}^{(k)} & 1 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \mu_n^{(k)} & 0 & 0 & 1 \end{pmatrix}$ В результате прямого хода метода Гаусса получим $A^{(n-1)} = U$,

$A = A^{(0)} = M_1^{-1} A^{(1)} = M_1^{-1} M_2^{-1} A^{(2)} = M_1^{-1} M_2^{-1} \dots M_{n-1}^{-1} A^{(n-1)}$,

где $A^{(n-1)} = U$ - верхняя треугольная матрица, а $L = M_1^{-1} M_2^{-1} \dots M_{n-1}^{-1}$ - нижняя треугольная матрица, имеющая вид $L = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ \mu_2^{(1)} & 1 & 0 & 0 & 0 & 0 \\ \mu_3^{(1)} & \mu_3^{(2)} & 1 & 0 & 0 & 0 \\ \dots & \dots & \dots \mu_{k+1}^{(k)} & 1 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ \mu_n^{(1)} & \mu_n^{(2)} & \mu_n^{(k)} & \mu_n^{(k+1)} & \dots & \mu_n^{(n-1)} & 1 \end{pmatrix}$.

Таким образом, искомое разложение $A = LU$ получено.

Параллельность. Предположим вначале, что мы располагаем вычислительной системой с локальной памятью и числом процессоров *p=n*. Пусть *i*-я строка матрицы A хранится в процессоре *i*. Тогда один из возможных вариантов организации *LU*-разложения заключается в следующем: на первом шаге первая строка рассылается всем процессорам, после чего вычисления

$$l_{i1} = \frac{a_{i1}}{a_{11}}, a_{ij} = a_{ij} - l_{i1} \cdot a_{1j}, j = 2, \dots, n$$

могут выполняться параллельно процессорами $p_2, \dots, p_n$. На втором шаге вторая строка приведенной матрицы рассылается из процессора p_2 процессорам $p_3, \dots, p_n$, а затем проводятся параллельные вычисления и т. д.

Отметим два главных недостатка этого подхода: значительный объем обмена данными между процессорами и уменьшение числа активных процессоров на 1 через каждый шаг.

Слоистая схема хранения. В более реалистической ситуации, когда $p \leq n$, проблема балансировки нагрузки в известной степени смягчается. Предположим, что $n = kp$ и применяется хранение по строкам. Поместим первые *k* строк матрицы *A* в память процессора 1, следующие *k* строк в память процессора 2 и т. д. Этот способ хранения назовем *блочной схемой.* Снова первая строка рассылается из процессора 1 остальным процессорам, а затем выполняются необходимые вычисления, однако теперь это делается блоками по k наборов операций в каждом процессоре. Как и прежде, в ходе приведения все большее число процессоров становятся бездействующими, однако отношение общего времени вычислений к времени обменов и времени простоев является возрастающей функцией от *k*.

Более привлекательна схема хранения, в которой строки, распределенные в разные процессоры, как бы прослаивают друг друга. Будем по-прежнему считать, что $n = kp$, и пусть строки *1, p+1, 2p+1,...* хранятся в процессоре 1, строки *2, p+2, 2p+2, ...* - в процессоре 2 и т. д. Такой способ хранения мы будем называть *циклической слоистой схемой.* Проблема простоя процессоров для этой схемы теряет остроту; например, процессор 1 будет работать почти до самого конца приведения, а именно до тех пор, пока не закончится обработка строки *(k-1)p+1*.

30. Технологии параллельного программирования. Программные средства реализации параллельных вычислений на многопроцессорных ЭВМ.

Стадии:

1. **Проектирование.** На данной стадии разработки могут применяться известные парадигмы и модели программирования, осуществляться привязка будущей программы к аппаратной платформе. Т.е. происходит декомпозиция задачи с учетом имеющихся аппаратных и программных средств.
2. **Разработка.** В качестве инструментов разработки могут использоваться:
   - **a.** Интерфейсы 2. Стандарты 3. параллельные расширения популярных языков
   - **b.** параллельные языки 5. специализированные библиотеки
   - **c.** средства создания и проектирования параллельных программ
   - **d.** специализированные прикладные пакеты.
3. **Отладка и мониторинг.** При этом используются средства для анализа и мониторинга производительности параллельных программ.
4. **Оптимизация.** Применяется комплексный подход к анализу эффективности программ для параллельных вычислительных систем.

На практике существуют несколько технологических подходов к программированию для параллельных вычислительных систем:

1. Программирование на стандартных и широко распространённых языках программирования с использованием ***высокоуровневых коммуникационных библиотек и интерфейсов (API)*** для организации межпроцессорного взаимодействия

   a. **MPI** (Message Passing Interface) - хорошо стандартизованный механизм для построения программ по модели обмена сообщениями. Существуют стандартные "привязки" MPI к языкам C, C++, Fortran 77, Fortran 90. Существуют бесплатные и коммерческие реализации почти для всех суперкомпьютерных платформ, а также для сетей рабочих станций UNIX и Windows NT. В настоящее время MPI - наиболее широко используемый и динамично развивающийся интерфейс из своего класса. **MPICH – это реализация интерфейса MPI**

   b. **OpenMP** - программный интерфейс (API) для программирования компьютеров с разделяемой памятью (SMP/NUMA). OpenMP можно использовать для программирования на языках Fortran и C/C++. До появления OpenMP не было подходящего стандарта для эффективного программирования на SMP-системах. OpenMP идеально подходит для разработчиков, желающих быстро распараллелить свои вычислительные программы с большими параллельными циклами. Разработчик не создает новую параллельную программу, а просто последовательно добавляет в текст последовательной программы OpenMP-директивы

   c. **PVM** (Parallel Virtual Machine) - общедоступная библиотека, предоставляющая возможности управления процессами с помощью механизма передачи сообщений. Существуют реализации PVM для самых различных платформ

   d. **BLACS** (Basic Linear Algebra Communication Subprograms) - библиотека передачи сообщений, ориентированная на решение задач линейной алгебры. BLACS используется в качестве коммуникационного интерфейса в библиотеке ScaLAPACK. Доступны версии, реализованные через MPI, MPL, NX, PVM

   e. **MPL** - библиотека передачи сообщений, разработанная корпорацией IBM. Используется на массивно-параллельных компьютерах RS/6000 SP и кластерах на базе систем RS/6000. Библиотечные функции могут использоваться в программах на языках C и Fortran. В настоящее время библиотека MPL вытеснена высокопроизводительной реализацией стандартного интерфейса MPI

   f. **Pthreads -** POSIX-интерфейс для организации нитей; поддерживается широко (практически на всех UNIX-системах), однако по многим причинам не подходит для практического параллельного программирования: 1. нет поддержки Fortran-а; 2. слишком низкий уровень; 3. нет поддержки параллелизма по данным; 4. механизм нитей изначально разрабатывался не для целей организации параллелизма

2. Введение специальных "распараллеливающих" конструкций в язык программирования. При этом могут создаваться оригинальные ***параллельные языки или параллельные расширения*** существующих языков (с сохранением преемственности)

    a. **Linda** - параллельный язык программирования. Программа рассматривается как совокупность процессов, которые могут обмениваться данными через пространство кортежей. В чистом виде практически не встречается, чаще всего используется совместно с другими языками высокого уровня как средство общения параллельных процессов.
    Из лекций Олега:
    Модель обмена сообщениями, базирующаяся на модели общей ассоциативной памяти. Есть общее хранилище для всех процессов, которое содержит кортежи («tes», 4.5, 5). Доступ осуществляется по содержанию кортежа. На запрос («tes», float a, 5) мы получим a = 4.5.

3. Использование средств ***автоматического распараллеливания*** последовательных программ (средств распознавания параллелизма в алгоритмах, средств автоматического и полуавтоматического распараллеливания последовательных программ)

    a. **BERT 77** - средство автоматического распараллеливания Fortran-программ, с генерацией параллельного кода в модели обмена сообщениями (PVM или MPI)
    b. **PIPS Workbench** - среда автоматического анализа и преобразования вычислительных Fortran-программ, с возможностью генерации OpenMP-директив, или программ в модели передачи сообщений (MPI/PVM)
    c. Программирование на стандартных языках. Использование в качестве конструктивных элементов заранее распараллеленных процедур из ***специализированных библиотек***
    d. **ATLAS** (Automatically Tuned Linear Algebra Software) - библиотека, позволяющая автоматически генерировать и оптимизировать численное программное обеспечение для процессоров с многоуровневой организацией памяти и конвейерными функциональными устройствами. Базируется на BLAS 3 уровня (Level 3). ATLAS требует некоторого времени для изучения основных параметров архитектуры целевого компьютера, а затем на основе этих параметров получает "оптимальный" код
    e. **PBLAS -** Параллельные версии базовых процедур линейной алгебры (BLAS), уровней 1, 2, 3. Библиотека разработана в рамках проекта ScaLAPACK
    f. Библиотека **ScaLAPACK** включает подмножество процедур LAPACK, переработанных для использования на MPP-компьютерах, включая: решение систем линейных уравнений, обращение матриц, ортогональные преобразования, поиск собственных значений и др. Разработана с использованием PBLAS и коммуникационной библиотеки BLACS

4. Использование инструментальных систем, облегчающих ***создание и проектирование*** параллельных программ

    a. **CODE** - графическая система создания параллельных программ. Параллельная программа представляется в виде графа, вершинами которого являются последовательные участки, а дуги соответствуют пересылкам данных. Последовательные участки могут быть написаны на любом языке, для пересылок используется PVM или MPI

5. Использование ***специализированных прикладных пакетов***

    a. **ABAQUS** – задачи инженерного анализа, прочности, теплофизики, деформации, упругости, пластичности, электромагнетизма
    b. **ANSYS** – аналогично ABAQUS
    c. **CFX** – задачи аэро- и гидродинамики, механики жидкостей и газов, горения и детонации

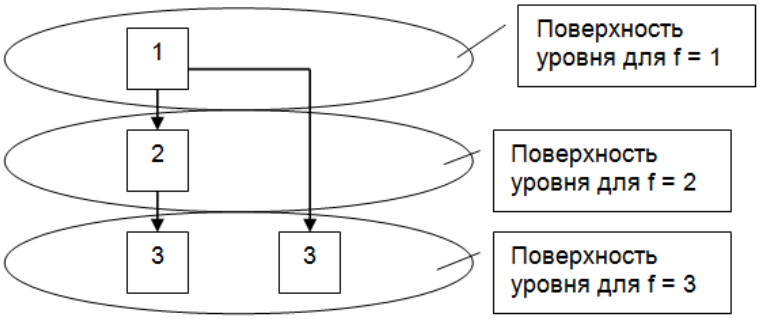
31. Эквивалентные преобразования программ в параллельном программировании.

Пара слов о графовых моделях программ/(с) Синявин/:

1. Граф управления: любая вершина – это оператор. Если программа допускает выполнение одного оператора после другого, то вершины соединяются дугой (пример: граф потоковых зависимостей)
2. Операционно-логическая история: у нас есть начальные исх данные для программы и мы наблюдаем за процессом ее выполнения. Вершина – срабатывание оператора.
3. Информационный граф: вершины – преобразователи, дугой соединяются те вершины, преобразователи которые имеют информационные зависимости.
4. История реализации программы: вершина – срабатывание оператора, дуга – передача информации.

Самый сложный с точки зрения построения – история реализации программы(он же решетчатый граф, он же граф алгоритма). ***Строгая развертка*** графа G это функционал f, для которого выполняется:f(v) > f(u), где u и v – вершины графа и из u в v имеется дуга. ***Обобщенная развертка*** графа G это функционал f, для которого выполняется: f(v) ≥ f(u), где u и v – вершины графа и из u в v имеется дуга.

Для чего собственно нужны развертки? Пусть известна какая-нибудь строгая развертка. Тогда: на любой ее поверхности уровня (поверхность уровня это совокупность всех точек, в которых функционал имеет одно и тоже значение (см. рисунок ниже):



никакие точки пространства итераций не могут быть связаны дугами графа G и даже не могут быть связаны путями графа). А следовательно: любые непересекающиеся множества точек любой поверхности уровня любой строгой развертки являются *параллельными по графу G* (можно выделить группы операций, которые можно выполнять параллельно, для этого требуется для программы построить граф, выделить там поверхности уровня и потом в этих поверхностях уровня выполнять параллельно какие-то операции).

Основные виды эквивалентных преобразований. Две программы будем называть эквивалентными, если они являются эквивалентными с точки зрения графов зависимостей: между графами зависимостей можно установить взаимно однозначное соответствие. Иными словами, последовательности вычислений, определяемые обоими графами, приводят к выполнению одних и тех же действий.

1. **Перестановка циклов** Преобразование состоит в перестановке местами каких-либо двух циклов в тесно вложенном гнезде циклов. Не ограничивая существенно общность, можно считать, что первый (самый внешний) цикл переставляется с *k*-ым.
2. **Слияние циклов** Рассмотрим две подряд идущие циклические конструкции с одинаковыми (с точностью до обозначения параметров) самыми внешними циклами. Преобразование заключается в слиянии этих конструкций в одну. Она представляет прежний внешний цикл, тело которого образовано из подряд идущих тел внешних циклов исходных конструкций.
3. **Переупорядочивание циклов** Преобразование состоит в выполнении последовательности перестановок местами пар операторов или циклов в пределах ближайшего объемлющего их цикла.
4. **Распределение цикла** Это преобразование является обратным по отношению к преобразованию слияния циклов.
5. **Скашивание цикла** … выполнение преобразований типа: a*b + c*b → (a+c) * b.
6. **Расщепление пространства итераций**

Преобразование заключается в представлении цикла в виде двух подряд идущих циклов. Тела обоих циклов совпадают с телом преобразуемого цикла. Нижняя (верхняя) граница изменения параметра первого (второго) цикла также совпадает с соответствующей границей исходного цикла. Поэтому нужно найти лишь верхнюю (нижнюю) границу первого (второго) цикла. Эти границы должны удовлетворять следующим условиям: нижняя граница второго цикла должна быть строго больше верхней границы первого цикла и обе границы должны находиться между границами преобразуемого цикла. Во всем остальном выбор границ произволен. Очевидно, что достаточно найти только верхнюю границу первого цикла.

7. **Выполнение операций цикла в обратном порядке** Формально преобразование заключается в перемене местами верхних и нижних границ изменения параметра какого-нибудь цикла программы и замене шага изменения параметра этого цикла с + 1 на - 1.
8. **Выделение стандартных операций**

Преобразование состоит в поиске в тексте программы одинаковых фрагментов и подстановки на их места обращений к оптимизированной процедуре.

32. Основные дифференциальные уравнения матфизики: уравнение колебаний, уравнение диффузии, уравнение переноса, уравнения газо- и гидродинамики, уравнение Максвелла, уравнение Шредингера, уравнение Клейна-Гордона-Фока и уравнение Дирака.

Дифференциальным уравнением в частных производных называется уравнение относительно неизвестной функции нескольких переменных, ее аргументов и ее частных производных различных порядков.

$$F\left(x_1, x_2, \ldots, x_n, \frac{\partial u}{\partial x_1}, \frac{\partial u}{\partial x_2}, \ldots, \frac{\partial u}{\partial x_n}, \ldots, \frac{\partial^k u}{\partial x_1^{k_1} \ldots \partial x_n^{k_n}}\right) = 0$$

Ур мат физики – дифуры в частных производных 2-го порядка. Линейные ДУ – которые содержат неизвестную функцию и ее производную только в первой степени.

Уравнение колебаний (волновое): задача Коши с краевыми условиями(смешанная задача)

$$\begin{cases} a^2 * u_{xx} = u_{tt} \ (1.1) \\ u(x,0) = \varphi(x) \quad (1.2) \\ u_t(x,0) = \psi(x) \ (1.3) \end{cases}$$

Краевые условия: u(0, t) = 0, u(l, t) = 0 (0<x<l). (1.2) и (1.3) граничные условия.

Ур-е (1.1) имеет решение вида: $u(x,t) = F(x-at) + \Phi(x+at) \quad (1.4)$

F и Ф – произвольные функции. Общее решение $u(x,t) = \frac{\varphi(x-at)+\varphi(x+at)}{2} + \frac{1}{2a}\int_{x-at}^{x+at} \psi(y)dy$ - формула Даламбера. Если есть краевые условия, то решение – это ряд Фурье:

$$u(x,t) = \sum_{k=1}^{\infty}\left(A_k \cos\frac{ak\pi}{l}t + B_k \sin\frac{ak\pi}{l}t\right)\sin\frac{k\pi}{l}x, \quad A_k = \frac{2}{l}\int_0^l \varphi(x)\sin\frac{\pi kx}{l}dx, B_k = \frac{2}{a\pi k}\int_0^l \psi(x)\sin\frac{\pi kx}{l}dx$$

Уравнение диффузии (теплопроводности): однородное с однородными гран условиями

$$\begin{aligned} &u_t = a^2 u_{xx}, \quad 0 < x < l,\ 0 < t \leqslant T \\ &u(x,\ 0) = \varphi(x); \quad 0 \leqslant x \leqslant l \\ &\left.\begin{aligned} u(0,\ t) = 0, \\ u(l,\ t) = 0. \end{aligned}\right\} \quad 0 \leqslant t \leqslant T \end{aligned}$$

Решение – ряд Фурье:

$$u(x,\ t) = \sum_{n=1}^{\infty} u_n(x) = \sum_{n=1}^{\infty} C_n \sin\left(\frac{\pi n}{l}x\right)\exp\left(-a^2\left(\frac{\pi n}{l}\right)^2 t\right).$$

$$C_n = A_n = \frac{2}{l}\int_0^l \varphi(\xi)\sin\left(\frac{\pi n}{l}\xi\right)\,d\xi.$$

Уравнение переноса: уравнение в частных производных, описывающее перенос сохраняющейся скалярной величины в пространстве.

$\frac{\partial \psi}{\partial t} + \nabla \cdot \mathbf{F} = 0$ где ∇• — оператор дивергенции, а $\mathbf{F}$ — вектор потока скалярной величины. Он равен произведению величины на скорость: $\psi\mathbf{u}$. Часто предполагается, что поле скоростей соленоидально, то есть $\nabla \cdot \mathbf{u} = 0$. В этом случае уравнение принимает вид:

$\frac{\partial \psi}{\partial t} + \mathbf{u} \cdot \nabla\psi = 0.$ В одномерной постановке имеет вид: $\frac{\partial \psi}{\partial t} + u\frac{\partial \psi}{\partial x} = 0.$

И при постоянном значении *u* имеет аналитическое решение: ψ(*x*,*t*) = ψ$_0$(*x* − *ut*)
где ψ$_0$ — произвольная гладкая (дифференцируемая) функция.

Уравнение Лапласа:

$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$ или $\triangle u = 0$. Решается как частный случай уравнения Бесселя,сперва записывается в полярных координатах r, φ :

$\frac{1}{r}\frac{\partial}{\partial r}\left(r\frac{\partial u}{\partial r}\right) + \frac{1}{r^2}\frac{\partial^2 u}{\partial \phi^2} = 0$,потом u = R*Φ, в результате решение выглядит так:

$u_k = (A_k \cos k\varphi + B_k \sin k\varphi)(C_k r^k + D_k r^{-k})$, где C и D - это из таблиц функции Бесселя

$$A_n = \frac{1}{\pi R^n}\int_{-\pi}^{\pi} f(t)\cos ntdt \quad B_n = \frac{1}{\pi R^n}\int_{-\pi}^{\pi} f(t)\sin ntdt$$

Уравнение Бернулли (гидродинамика): Обыкновенное дифференциальное уравнение вида:

$y' + P(x)y = Q(x)y^n, \quad n \neq 0,\ 1,$

Решение: Заменим $y = uv$. Тогда $\dot{u}v + u(\dot{v} + a(x)v) = b(x)(uv)^n$.

Подберем $v(x) \not\equiv 0$ так, чтобы было $\dot{v} + a(x)v = 0$

для этого достаточно решить уравнение с разделяющимися переменными 1-го порядка. После этого для определения u получаем уравнение $\frac{\dot{u}}{u^n} = b(x)v^{n-1}$ — уравнение с разделяющимися переменными.

Уравнение Максвелла: описывают электромагнитное поле

$$\mathrm{rot\ H} = \varepsilon * \frac{\mathrm{dE}}{\mathrm{dt}} \qquad \mathrm{rot\ E} = \mu * \frac{\mathrm{dH}}{\mathrm{dt}}$$

Где H - напряженность магнитного поля; E- напряженность электрического поля;
ε - диэлектрическая проницаемость; μ - магнитная проницаемость.

Уравнение Шредингера: комплекснозначная функция Ψ, описывающая чистое состояние объекта, называется волновой функцией. Эта функция связана с вероятностью обнаружения объекта в одном из чистых состояний (квадрат модуля волновой функции представляет собой плотность вероятности). Пусть волновая функция задана в N-мерном пространстве, тогда в каждой точке с координатами $\vec{r}(x_1, x_2, x_3, \ldots, x_n)$, в определенный момент времени *t* она будет иметь вид $\Psi(\vec{r}, t)$. В таком случае уравнение Шрёдингера запишется в виде:

$$-\frac{\hbar^2}{2m}\nabla^2\Psi + E_p\Psi = i\hbar\frac{\partial}{\partial t}\Psi$$ где $\hbar = \frac{h}{2\pi}$, h — постоянная Планка; m — масса частицы, $E_p(\vec{r})$ — внешняя по отношению к частице потенциальная энергия в точке $\vec{r}(x_1, x_2, x_3, \ldots, x_n)$.

Уравнения Клейна — Гордона — Фока является релятивистской версией уравнения Шрёдингера. Для свободной частицы записывается в виде

$$\nabla^2\psi - \frac{1}{c^2}\frac{\partial^2}{\partial t^2}\psi = \frac{m^2c^2}{\hbar^2}\psi$$

Уравнение Дирака — квантовое уравнение движения электрона, удовлетворяющее требованиям теории относительности. Уравнение Дирака записывается в виде

$$\left(\alpha_0 mc^2 + \sum_{j=1}^{3}\alpha_j p_j\, c\right)\psi(\mathbf{x}, t) = i\hbar\frac{\partial\psi}{\partial t}(\mathbf{x}, t)$$

где *m* — масса покоя электрона, *c* — скорость света, *p* — оператор импульса, $\hbar$— постоянная Планка, **x** и *t* пространственные и временная компонента соответственно, и *ψ*(**x**, *t*) — четырёхкомпонентная волновая функция (биспинор).
α_j — линейные операторы, которые действуют на волновую функцию

33. Общая, каноническая и основная задача линейного программирования. Графическое решение задачи. Симплекс-метод и симплекс-таблицы. Выбор начального допустимого решения. Двойственность в линейном программировании. Анализ на чувствительность.

Лин прог – теория и численные методы решения задач нахождения экстремума (максимума или минимума) линейной функции многих переменных при наличии линейных ограничений, т.е. линейных равенств или неравенств, связывающих эти переменные.

Общая задача линейного программирования (ОЗЛП): найти значения переменных $x_1, \dots, x_n$, максимизирующие линейную форму $f(x_1, \dots, x_n) = c_1x_1 + \dots + c_nx_n$ (1.4) при условиях:

$\sum_{j=1}^{n} a_{ij}x_j \le b_i\ ,\ i = 1 \dots m_1\ \ (m_1 \le m)$ (1.5)

$\sum_{j=1}^{n} a_{ij}x_j = b_i,\ \ i = m_1 + 1, \dots m$

$x_j \ge 0,\ j = 1 \dots p\ \ (p \le n)$ (1.6)

Соотношения (1.5) и (1.6) будем называть соответственно функциональными и прямыми ограничениями ЗЛП. Значения переменных $x_j\ (j = 1 \dots n)$ - компоненты некоторого вектора $\bar{X} = (X_1, \dots, X_n)$ пространства E_n.

Планом или допустимым решением ЗЛП наз вектор $\bar{X}$ пространства E_n, компоненты кот. удовлетворяют (1.5) и (1.6). Множество всех планов задачи лин прога (1.4)-(1.6) обозначается Р.

План $\bar{X}^* = (X_1^*, \dots, X_n^*)$ наз **решением** ЗЛП или ее **оптимальным планом**, если $f(\bar{X}^*) = \max_{\bar{X} \in P} f(\bar{X})$. Будем говорить, что ЗЛП разрешима, если она имеет хотя бы один оптимальный план.

Основная задача: найти значения переменных $X_1, \dots, X_n$ максимизирующие линейную форму $f(\bar{X}) = \sum_{j=1}^{n} c_{ij}x_j$ (1.7) при условиях $\sum_{j=1}^{n} a_{ij}x_j \le b_i\ ,\ i = 1 \dots m$ (1.8) или в векторно-матричной форме

$$\begin{cases} f(\bar{x}) = (\bar{c}, \bar{x}) \to \max & (1.10) \\ A\bar{x} \le \bar{b} & (1.11) \\ \bar{x} \ge \bar{0} & (1.12) \end{cases}$$

, где $\bar{C} = (C_1 \dots C_n),\ \bar{b} = (\overline{b_1}, \dots, \overline{b_m}),\ A = (a_{ij})$ - матрица коэффициентов ограничений (1.8). Задача (1.7)-(1.9) или (1.10)-(1.12) наз основной ЗЛП. Основная ЗЛП является частным случаем общей ЗЛП при $m_1 = m,\ p = n$.

Каноническая задача: Для построения общего метода решения ЗЛП разные формы ЗЛП должны быть приведены к канонической задаче линейного программирования (КЗЛП):

1) все функциональные ограничения записываются в виде равенств с неотрицательной правой частью; 2) все переменные неотрицательны; 3) целевая функция подлежит максимизации.

Таким образом, КЗЛП имеет вид в векторно-матричной форме

$$\begin{cases} f(\bar{x}) = (\bar{c}, \bar{x}) \to \max & (1.16) \\ A\bar{x} = \bar{b} & (1.17) \\ \bar{x} \ge \bar{0}\ ,\ \bar{b} \ge 0 & (1.18) \end{cases}$$

КЗЛП является частным случаем общей ЗЛП при $m_1 = 0,\ p = n$.

Любую ЗЛП можно привести к каноническому виду, используя следующие правила:

а) максимизация целевой функции $f(\bar{X}) = c_1x_1 + \dots + c_nx_n$ равносильна минимизации целевой функции $-f(\bar{X}) = -c_1x_1 - \dots - c_nx_n$;

б) ограничение в виде неравенства, например $3x_1 + 2x_2 - x_3 \le 6$, может быть приведено к стандартной форме $3x_1 + 2x_2 - x_3 + x_4 = 6$, где новая переменная x_4 неотрицательна. Ограничение $x_1 - x_2 + 3x_3 \ge 10$ может быть приведено к стандартной форме $x_1 - x_2 + 3x_3 - x_5 = 10$, где новая переменная x_5 неотрицательна;

в) если некоторая переменная X_K может принимать любые значения, а требуется, чтобы она была неотрицательная, ее можно привести к виду $X_K = X_K^{'} - X_K^{''}$, где $X_K^{'} \ge 0$ и $X_K^{''} \ge 0$

Графический метод решения: им целесообразно решать ЗЛП, содержащие не более двух переменных. Сначала на координатной плоскости x1Ox2 строится допустимая многоугольная область (область допустимых решений, область определения), соответствующая ограничениям:

$a_{11}x_1 + a_{12}x_2 \le b_1$

…

$a_{m1}x_1 + a_{m2}x_2 \le b_m$

$x_1 \ge 0,\ \ x_2 \ge 0$

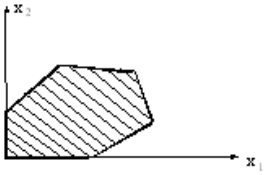


Рис. 6

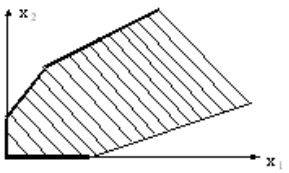
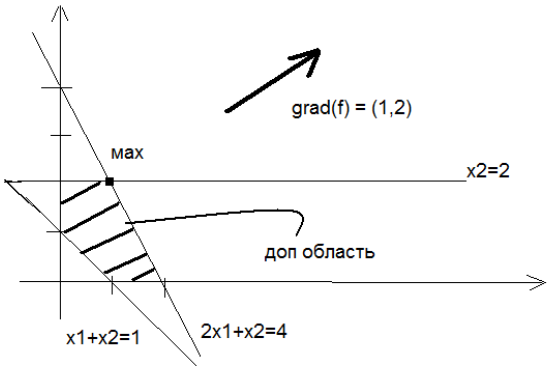


Рис. 7

1. Основной случай - получающаяся область имеет вид ограниченного выпуклого многоугольника (рис. 6)).
2. Неосновной случай - получается неограниченный выпуклый многоугольник, имеющий вид, подобный изображенному на рис. 7.
3. Наконец, возможен случай, когда неравенства противоречат друг другу, и допустимая область вообще пуста.



Пример из лекций ((с) Грибов). $F = x_1 + 2x_2\ Fmax - ?$
Ограничения: $2x_1 + x_2 \le 4;\ -x_1 - x_2 \le -1;\ x_2 \le 2;\ x_1 \ge 0; x_2 \ge 0$
Строим график с линиями уровня, указываем направление градиента: $Fmax = f(1,2) = 5$

Симплекс-метод и симплекс-таблицы.
Рассмотрим каноническую задачу линейного программирования (КЗЛП)
$\max(\bar{c}, \bar{x})$ (1.22)
$\sum_{j=1}^{n} \bar{a}_j x_j = \bar{b}$ (1.23) где $\bar{a}_j$ - j-ый столбец матрицы А.
$\bar{x} \ge \bar{0},\ \bar{b} \ge \bar{0}$ (1.24)
Будем в дальнейшем считать, что ранг матрицы А системы уравнений $A\bar{x} = \bar{b}$ равен m, причем $m < n$. **Опорным планом** (ОП) ЗЛП наз план, являющийся базисным решением системы линейных уравнений $A\bar{x} = \bar{b}$. $rang(A) = m \to m$ компонент базисного решения системы линейных уравнений $A\bar{x} = \bar{b}$, являющихся значениями соответствующих ему базисных переменных, наз базисными компонентами этого решения (они неотрицательны, остальные $n - m$ компонент $= 0$).
К-матрицей КЗЛП наз расширенную матрицу системы линейных уравнений, равносильной системе $\sum_{j=1}^{n} \bar{a}_j x_j = \bar{b}$, содержащую единичную подматрицу на месте первых n своих столбцов и все элементы $(n+1)$-го столбца которой неотрицательны (в обозн $N = (N_1, \dots, N_m)$ – вектор номеров базисных(единичных) столбцов матрицы K, а $X_N = (b_1, \dots, b_m)$ – вектор базисных компонент опорного плана, определяемого матрицей К, отличных от нуля). Число -матриц конечно и не превышает C_n^m. Каждая К-матрица определяет ОП КЗЛП и наоборот.
АЛГОРИТМ СИМПЛЕКС-МЕТОДА. Известна исходная К-матрица $K^{(0)}$ задачи линейного программирования, определяющая исходный опорный план. $\bar{X}_{\bar{N}^{(0)}} = (b_1^{(0)}, b_2^{(0)}, \dots, b_m^{(0)})$
$\bar{N}^{(0)} = (N_1^{(0)}, N_2^{(0)}, \dots, N_m^{(0)})$
В симплексном методе последовательно строят К-матрицы $K^{(0)} \dots K^{(S)}$ задачи линейного программирования, пока не выполнится критерий оптимальности или критерий, позволяющий убедиться в отсутствии конечного решения. Рассмотрим алгоритм S-й итерации симплексного метода. В начале S-й итерации имеем К-матрицу $K^{(S-1)}$ задачи линейного программирования, определяющую опорный план
$\bar{X}_{\bar{N}^{(s-1)}} = (b_1^{(s-1)}, b_2^{(s-1)}, \dots, b_m^{(s-1)})$
$\bar{N}^{(s-1)} = (N_1^{(s-1)}, N_2^{(s-1)}, \dots, N_m^{(s-1)})$
Шаг 1. Вычисляем для столбцов $\bar{a}_j^{(s-1)}$ матрицы $K^{(s-1)}, (j \ne N_i^{(s-1)}, i = \overline{1,m})$ симплекс-разности $\Delta_j^{(s-1)}$ и находим номер K из условия $\Delta_k^{(s-1)} = \min \Delta_j^{(s-1)}, 1 \le j \le n$. Величина $\Delta_k^{(s)} = \left(\bar{C}_{\bar{N}^{(s)}}, \bar{a}_j^{(s)}\right) - C_j$, где $\bar{C}_{\bar{N}^{(s)}}$ - вектор коэффициентов лин ф-ции $f(\bar{x}) = (\bar{c}, \bar{x})$ при базисных $(\bar{N}^{(s)})$ переменных опорного плана, опр-го матр $K^{(S)}$ – j-я симплекс-разность матр $K^{(S)}$.
Шаг 2. Если $\Delta_k^{(s-1)} \ge 0$, то опорный план $\bar{X}_{\bar{N}^{(s-1)}}$ является оптимальным, а $f\left(\bar{X}_{\bar{N}^{(s-1)}}\right) = (\bar{C}_{\bar{N}^{(s-1)}}, \bar{X}_{\bar{N}^{(s-1)}})$ - оптимальное значение линейной формы $f(\bar{X})$, иначе переходим к шагу 3.
Шаг 3. Если $a_{ik}^{(s-1)} \le 0,\ i = \overline{1,m}$, то ЗЛП не имеет конечного решения, иначе находим номер l из условия $\theta^{(s-1)} = \min_{1 \le i \le m}\left(b_i^{(s-1)} / a_{ik}^{(s-1)}\right) = b_l^{(s-1)} / a_{lk}^{(s-1)}$ направляющий элемент на S-й итерации метода есть элемент $a_{lk}^{(s-1)}$.
Шаг 4. Вычисляем компоненты вектора $\bar{N}^{(s)}$: $N_i^{(s)} = N_i^{(s-1)}, i \ne l, N_l^{(s)} = k???$
Шаг 5. Производим один шаг метода Жордана-Гаусса с направляющим элементом $a_{lk}^{(s-1)}$. Присваиваем переменной S алгоритма значение S+1 и переходим к шагу 1.

Симплекс-таблица - содержит промежуточные результаты вычислений:

S	i	$\bar{N}^{(s)}$	$\bar{C}_{\bar{N}^{(s)}}$	$\bar{X}_{\bar{N}^{(s)}} = \bar{b}^{(s)}$	3 $\bar{a}_1^{(s)}$	2 $\bar{a}_2^{(s)}$	0 $\bar{a}_3^{(s)}$	0 $\bar{a}_4^{(s)}$	0 $\bar{a}_5^{(s)}$	0 $\bar{a}_6^{(s)}$	$\theta^{(S)}$
	1	3	0	6	1	2	1	0	0	0	6
	2	4	0	8	2	1	0	1	0	0	4
0	3	5	0	1	-1	1	0	0	1	0	-
	4	6	0	2	0	1	0	0	0	1	-
	5	$\Delta_j^{(0)}$		f=0	-3	-2	0	0	0	0	k=1 l=2
	1	3	0	2	0	3/2	1	-1/2	0	0	4/3
	2	1	3	4	1	1/2	0	1/2	0	0	8
1	3	5	0	5	0	3/2	0	1/2	1	0	10/3
	4	6	0	2	0	1	0	0	0	1	2

Двойственность в линейном программировании: Двойственная задача – это вспомогательная задача линпрога, формулируемая с помощью определенных правил непосредственно из условий исходной, или прямой задачи, которая применима к любой форме представления прямой задачи. В основу такого подхода положен тот факт, что использование симплекс-метода требует приведения любой ЗЛП к каноническому виду. Пусть прямая задача записана в каноническом виде:

$$\max(\min) f(\bar{x}) = \sum_{j=1} c_j x_j \qquad (1.47)$$

$$\sum_{j=1}^{n} a_{ij} x_j = b_i, i = 1...m \qquad (1.48)$$

$$x_j \geq 0, j = 1...n \qquad (1.49)$$

Задачей, двойственной к ЗЛП (1.47)-(1.49), называется следующая ЗЛП

$$\min(\max) g(\bar{y}) = \sum_{i=1}^{m} y_i b_i \qquad (1.50)$$

$$\sum_{i=1}^{m} y_i a_{ij} \geq (\leq) c_j, j = 1...n \qquad (1.51)$$

y_i не ограничены в знаке, i = 1...m (1.52)

двойственная ЗЛП строится по следующим правилам:

1) Каждому ограничению прямой задачи соответствует переменная двойственной задачи, т.е. число переменных двойственной задачи *(y1,..., ym)* равно числу ограничений прямой задачи.

2) Каждой переменной прямой задачи соответствует ограничение двойственной задачи, т.е. число ограничений двойственной задачи равно числу переменных прямой задачи.

3) Матрица функциональных ограничений двойственной задачи получается путем транспонирования матрицы функциональных ограничений прямой задачи.

4) Вектор *C* целевой функции прямой задачи становится вектором правой части ограничений двойственной задачи, а вектор *b* правой части прямой задачи – вектором целевой функции двойственной задачи.

5) Если целевая функция прямой задачи максимизируется, то целевая функция двойственной задачи минимизируется, а ограничения имеют вид ≥ , и наоборот.

Прямая задача

$$\max f(\bar{x}) = (\bar{C}, \bar{x})$$

$$P = \begin{cases} A\bar{x} = \bar{b} \\ \bar{x} \geq \bar{0} \end{cases}$$

Двойственная задача

$$\min g(\bar{y}) = (\bar{y}, \bar{b})$$

$$Q = \begin{cases} \bar{y}A \geq \bar{C} \\ y \text{ – не ограничен в знаке} \end{cases} \qquad (1.53)$$

$$\min f(\bar{x}) = (\bar{C}, \bar{x})$$

$$P = \begin{cases} A\bar{x} = \bar{b} \\ \bar{x} \geq \bar{0} \end{cases}$$

$$\max g(\bar{y}) = (\bar{y}, \bar{b})$$

$$Q = \begin{cases} \bar{y}A \leq \bar{C} \\ y \text{ – не ограничен в знаке} \end{cases} \qquad (1.54)$$

34. Общая задача нелинейного программирования. Выпуклые множества и выпуклые функции. Задача выпуклого программирования. Условия оптимальности в задачах выпуклого программирования. Методы условной оптимизации. Методы штрафных и барьерных функций. Методы проекции градиента, приведенного градиента, условного градиента. Метод проекции точки на множество.

В общем виде задача нелинейного программирования состоит в определении максимального (минимального) значения функции $f(x_1, x_2, \dots, x_n)$ при условии $\begin{cases} g_i(x_1, x_2, \dots, x_n) \le b_i,\ i = \overline{1,k} \\ g_i(x_1, x_2, \dots, x_n) = b_i,\ i = \overline{k+1,m} \end{cases}$, где f и g_i - некоторые известные функции *n* переменных, а b_i - заданные числа. Даже если область допустимых решений - выпуклая, то в ряде задач целевая функция может иметь несколько локальных экстремумов. С помощью большинства же вычислительных методов можно найти точку локального оптимума, но нельзя установить, является ли она точкой глобального (абсолютного) оптимума или нет. Если задача содержит нелинейные ограничения, то область допустимых решений не является выпуклой, и кроме глобального оптимума могут существовать точки локального оптимума.

ВЫПУКЛЫЕ Ф-ЦИИ И МН-ВА. Функция $f(x_1, x_2, \dots, x_n)$, заданная на выпуклом множестве *X* (содержит вместе с любыми двумя точками соединяющий их отрезок), наз **выпуклой**, если для любых двух точек $X^{(1)}$ и $X^{(2)}$ из *X* и любого $0 \le \lambda \le 1$ выполняется соотношение

$f\left[\lambda X^{(2)} + (1-\lambda)X^{(1)}\right] \le \lambda f\left(X^{(2)}\right) + (1-\lambda) f\left(X^{(1)}\right)$. (4)

Функция $f(x_1, x_2, \dots, x_n)$, заданная на выпуклом множестве *X*, наз **вогнутой**, если для любых двух точек $X^{(1)}$ и $X^{(2)}$ из *X* и любого $0 \le \lambda \le 1$ выполняется соотношение

$f\left[\lambda X^{(2)} + (1-\lambda)X^{(1)}\right] \ge \lambda f\left(X^{(2)}\right) + (1-\lambda) f\left(X^{(1)}\right)$. (5)

Если неравенства (4) и (5) считать строгими и они выполняются при $0 \le \lambda \le 1$, то функция $f(x_1, x_2, \dots, x_n)$ является строго выпуклой (строго вогнутой). Выпуклость и вогнутость функций определяется только относительно выпуклых множеств. Если $f(x) = \sum_{j=1}^{k} f_j(x)$, где $f_j(x)$, - выпуклые (вогнутые) функции на некотором выпуклом множестве $X \subset E^n$, то функция $f(x)$ - также выпуклая (вогнутая) на *X*.

ЗАДАЧА ВЫПУКЛОГО ПРОГРАММИРОВАНИЯ. Рассмотрим задачу нелинейного программирования $f(x_1, x_2, \dots, x_n) \to max$ **(6)** при ограничениях $g_i(x_1, x_2, \dots, x_n) = b_i,\ i = \overline{1,m}$, **(7)** и $x_j \ge 0, j = \overline{1,n}$ **(8)**

Для решения задачи в такой общей постановке не существует универсальных методов. Однако для отдельных классов задач, в которых сделаны дополнительные ограничения относительно свойств функций $f(x)$ и $g_i(x)$, разработаны эффективные методы их решения.

Говорят, что множество допустимых решений задачи (6) - (8) удовлетворяет условию **регулярности**, если существует, по крайней мере, одна точка $X^{(0)}$, принадлежащая области допустимых решений такая, что $g_i(X^{(0)}) < b_i,\ i = \overline{1,m}$.

Задача (6) - (8) называется **задачей выпуклого программирования**, если функция $f(x_1, x_2, \dots, x_n)$ является вогнутой (выпуклой), а функции $g_i(x_1, x_2, \dots, x_n)$ $(i = \overline{1,m})$ - выпуклыми.

Функцией Лагранжа задачи выпуклого программирования (6) - (8) называется функция $L(x_1, x_2, \dots, x_n, y_1, y, \dots, y_m) = f(x_1, x_2, \dots, x_n) + \sum_{i=1}^{m} y_i [b_i - g_i(x_1, x_2, \dots, x_n)]$ где $y_1, y, \dots, y_m$ - множители Лагранжа.

Точка $(X^{(0)}, Y^{(0)}) = (x_1^0, x_2^0, \dots, x_n^0, y_1^0, y_2^0, \dots, y_m^0)$ называется **седловой точкой функции Лагранжа,** если $L(x_1, x_2, \dots, x_n, y_1^0, y_2^0, \dots, y_m^0) \le L(x_1^0, x_2^0, \dots, x_n^0, y_1^0, y_2^0, \dots, y_m^0) \le L(x_1^0, x_2^0, \dots, x_n^0, y_1, y, \dots, y_m)$ для всех $x_j \ge 0$ $(j = \overline{1,n})$ и $y_i \ge 0$ $(i = \overline{1,m})$.

к вопросу об условии оптимальности в задачах выпуклого программирования:

Теорема 1 (Куна - Таккера): Для задачи выпуклого программирования (6) - (8), множество допустимых решений которой обладает свойством регулярности, $X^{(0)} = (x_1^0, x_2^0, \dots, x_n^0)$ является оптимальным решением тогда и только тогда, когда существует такой вектор $Y^{(0)} = (y_1^0, y_2^0, \dots, y_m^0)$ $(y_i^0 \ge 0, i = \overline{1,m})$, что $(X^{(0)}, Y^{(0)})$ - седловая точка функции Лагранжа.

Если предположить, что функции *f* и g_i непрерывно дифференцируемы, то теорема Куна - Таккера может быть дополнена аналитическими выражениями, определяющими необходимые и достаточные условия того, чтобы точка $(X^{(0)}, Y^{(0)})$ была седловой точкой функции Лагранжа, т. е. являлась решением задачи выпуклого программирования:

$$\begin{cases} \partial L_0/\partial x_j \le 0,\ j=\overline{1,n} \\ x_j^0 \partial L_0/\partial x_j = 0,\ j=\overline{1,n} \\ x_j^0 \ge 0,\ j=\overline{1,n} \\ \partial L_0/\partial y_i,\ i=\overline{1,m} \\ y_i^0 \partial L_0/\partial y_i,\ i=\overline{1,m} \\ y_i^0 \ge 0,\ i=\overline{1,m} \end{cases}$$ где $\partial L_0/\partial x_j$ и $\partial L_0/\partial y_i$ значения соответствующих частных производных функции Лагранжа, вычисленных в седловой точке.

Методы условной оптимизации. В дальнейшем будем рассматривать следующую задачу: $f(\bar{x}) \to max$ на множестве $P = \{\bar{x} \in E_n : g_i(\bar{x}) \le 0, i = \overline{1,m}, x_j \ge 0, j = \overline{1,n}\}$ где $f(\bar{x})$ и $g_i(\bar{x})$ – нелинейные функции. При решении задач нелинейного программирования ввиду нелинейности функции $g_i(\bar{x})$ выпуклость допустимого множества решений Р и конечность числа его крайних точек (в отличие от ЗЛП) необязательны. Если задача имеет решение, то максимум $f(\bar{x})$ может достигаться в крайней точке Р, в одной из граничных точек или в точке, расположенной внутри Р.

Решением или точкой максимума задачи условной оптимизации наз такой вектор $\bar{x}^* \in P \subset E_n$, что $f(\bar{x}^*) \ge f(\bar{x})$ для всех $\bar{x} \in P$, т.е. $f(\bar{x}^*) = \max_{\bar{x} \in P} f(\bar{x})$

Вектор $\bar{S}_k$ наз возможным направлением подъема функции $f(\bar{x})$ в точке $\bar{X}_k \in P$, если существует такое действительное число $\beta_0 > 0$, что для всех $\beta \in (0, \beta_0)$: $(\bar{x}_k + \beta \bar{S}_k) \in P$ и $f(\bar{x}_k + \beta \bar{S}_k) > f(\bar{x}_k)$.

Метод решения задачи условной оптимизации - это итерационный процесс, в котором исходя из начальной точки $\bar{x}_0 \in P$, получают последовательность точек $\bar{x}_k \in P$, монотонно увеличивающих значения $f(\bar{x})$ (методы подъема). Элементы этой последовательности точек определяются следующим образом: $\bar{x}_{k+1} = \bar{x}_k + \beta_k \bar{S}_k$, где $\bar{S}_k$ - возможное направление подъема функции в точке $\bar{x}_k$. β_k находится при решении задачи одномерной оптимизации: $f(\bar{x}_k + \beta \bar{S}_k) \to max_\beta$. Прежде чем определять направление подъема функции $f(\bar{x})$ в точке $\bar{x}_k$, следует вычислить множество таких возможных направлений $\bar{S}_k$, для которых существовала бы окрестность точки $\bar{x}_k$, принадлежащая Р.

Метод штрафных функций: задача метода штрафных функций - преобразование задачи минимизации функции $z = f(x)$ с соответствующими ограничениями $c_j(x) > 0,\ j = 1,2,\dots,m$, в задачу поиска минимума без ограничений функции $X(x) = f(x) + P(x)$. Функция $P(x)$ является штрафной. Необходимо, чтобы при нарушении ограничений она «штрафовала» функцию Z, т.е. увеличивала её значение, следовательно минимум функции Z будет находится внутри области ограничений. Функция $P(x)$, удовлетворяющая этому условию, может быть не единственной. $P(x) = r \sum_{j=1}^{m} (1/c_j(x))$ где r – положительная величина. Тогда функция $Z = \varphi(x, r)$ принимает вид $Z = f(x) + r \sum_{j=1}^{m} (1/c_j(x))$. Полагая r достаточно малой величиной, для того чтобы влияние $P(x)$ было малым в точке минимума, мы можем сделать точку минимума функции $\varphi(x, r)$ без ограничений совпадающей с точкой минимума задачи с ограничениями.

Метод барьерных функций: Метод штрафных функций начинает работать с допустимой точки x_0 и генерирует последовательность допустимых точек $x_0, \dots, x_n$. Метод барьерных функций, наоборот, начинает поиск с недопустимой точки и генерирует последовательность недопустимых решений, которая приближается к оптимальному решению извне допустимой области. Пусть имеется задача минимизировать $f(x)$ при ограничениях $g_i(x) \ge 0, i = \overline{1,m};\ h_i(x) = 0, i = \overline{m+1,l}$, В частности, для искомых функций-ограничений целесообразно использовать барьерную функцию следующего вида: $\alpha(x) = \sum_{i=1}^{m} R_1\big(g_i(x)\big) + \sum_{i=m+1}^{l} R_2(h_i(x))$, R_1, R_2 - непрерывные функции, которые удовлетворяют условиям: $R_1(y) = 0$, если $y \ge 0$ и $R_1(y) > 0$, если $y < 0$, $R_2(y) = 0$, если $y = 0$ и $R_2(y) > 0$, если $y \ne 0$. Типичными являются следующие выражения для функций R_1, R_2: $R_1(y) = (\max\{0, -y\})^p$, $R_2 = |y|^p$, где р – целое положительное число.

Далее от исходной задачи переходим к задаче безусловной оптимизации вспомогательной функции: минимизировать $f(x) + r\alpha(x)$, где $r > 0$ - штрафной коэффициент. Пусть α – непрерывная функция. Обозначим $\theta(r) = \inf\{f(x) + r\alpha(x)\}$. Подход, связанный с барьерной функцией состоит в решении задачи вида: максимизировать $\theta(r)$ при ограничении $r \ge 0$.

Метод проекции градиента: если точка посл-ти $\{u_n\}$ находится внутри допустимого множества, то этот метод полностью совпадает с любым градиентным методом. Если же в процессе итераций, пользуясь градиентным методом, посл-ть выходит за допустимую область, ее, так называемым проектированием возвращают: проекция точки вне области на эту область – точка на границе, соответствующая кратчайшему расстоянию. Если область выпуклая, то проекция не существует. Метод возможен, если проекция легко вычисляется (например, линейная область).

Метод условного градиента: на каждой итерации заменяем функцию линейной и ищем минимум линейной функции, если у нее легко находится минимум.

35. Интерполяция и приближение. Постановка задачи. Интерполяционный многочлен Лагранжа. Разделенные разности и интерполяционная формула Ньютона. Интерполяционные сплайны. Погрешность интерполяционных формул.

Интерполяция— способ нахождения промежуточных значений величины по имеющемуся дискретному набору известных значений. Часто приходится оперировать наборами значений, полученных экспериментальным путём или методом случайной выборки. Как правило, на основании этих наборов требуется построить функцию, на которую могли бы с высокой точностью попадать другие получаемые значения. Такая задача называется аппроксимацией кривой. Интерполяцией называют такую разновидность аппроксимации, при которой кривая построенной функции проходит точно через имеющиеся точки данных. Существует также близкая к интерполяции задача, которая заключается в аппроксимации какой-либо сложной функции другой, более простой функцией.Рассмотрим систему несовпадающих точек x_i ($i \in 0, 1, \ldots, N$) из некоторой области D. Пусть значения функции f известны только в этих точках: $y_i = f(x_i), \quad i = 1, \ldots, N.$

Задача интерполяции состоит в поиске такой функции F из заданного класса функций, что $F(x_i) = y_i, \quad i = 1, \ldots, N.$ Точки x_i называют узлами интерполяции, а их совокупность — интерполяционной сеткой. Пары (x_i, y_i) называют точками данных или базовыми точками.

Разность между «соседними» значениями $\Delta x_i = x_i - x_{i-1}$ — шагом интерполяционной сетки. Он может быть как переменным так и постоянным. Функцию $F(x)$ — интерполирующей функцией . На практике чаще всего применяют интерполяцию многочленами. Это связано прежде всего с тем, что многочлены легко вычислять, легко аналитически находить их производные и множество многочленов плотно в пространстве непрерывных функций (теорема Вейерштрасса).

Пусть нам даны значения функции f(x) в n+1 точке: yk = f(xk) при k = 0, 1, … n.

Тогда существует полином L(x) = A0 + A1x + … + Anxn, значения которого совпадают со значениями f(x) во всех точках xk. Этот полином называется полиномом Лагранжа и служит примером глобальной интерполяции.Коэффициенты Ak можно найти, решив следующую систему:

$$\begin{cases} A_0 + A_1 x_0 + \ldots + A_n x_0^n = y_0 \\ A_0 + A_1 x_1 + \ldots + A_n x_1^n = y_1 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ A_0 + A_1 x_n + \ldots + A_n x_n^n = y_n \end{cases}$$

Определитель этой системы является определителем Вандермонда. Он не равен нулю, поэтому система всегда имеет единственное решение.

Коэффициенты Ak можно найти, не решая систему, по следующей формуле:

$$L(x) = \sum_{k=0}^{n} y_k Q_k(x), \quad (3)$$

где Qk(x) - полиномы n-ой степени, называемые лагранжевыми коэффициентами и имеющие вид:

$$Q_k(x) = \frac{(x - x_0) \cdot (x - x_1) \ldots (x - x_{k-1}) \cdot (x - x_{k+1}) \ldots (x - x_n)}{(x_k - x_0) \cdot (x_k - x_1) \ldots (x_k - x_{k-1}) \cdot (x_k - x_{k+1}) \ldots (x_k - x_n)}$$

Действительно, степень каждого полинома Qk в точности равна n, сам он равен 1 в точке x = xk и обращается в нуль в остальных узлах интерполяции. Формула (3) позволяет вычислять значения полинома Лагранжа и без нахождения его коэффициентов.Погрешность интерполяции функции полиномом Лагранжа на отрезке [a,b]=[x0,xn] оценивается формулой:(4)

$$|f(x) - L(x)| \le \frac{M_{n+1}}{(n+1)!} \cdot |(x - x_0) \cdot (x - x_1) \ldots (x - x_n)|, \quad \text{где} \quad M_{n+1} = \max_{x \in [a,b]} |f^{(n+1)}(x)|.$$

Разделенные разности нулевого порядка совпадают со значениями самой функции. Разделенные разности первого порядка: $f(x_i, x_j, x_l) = \frac{f(x_i, x_j) - f(x_j, x_l)}{x_i - x_l}$.Разделенные разности k- го порядка : $f(x_1, \ldots, x_{k+1}) = \frac{f(x_1, \ldots, x_k) - f(x_2, \ldots, x_{k+1})}{x_1 - x_{k+1}}$ (14). Пусть P_n (x) многочлен степени n. Разность P_n (x) – Pn (x0) обращается в нуль при x=x0, следовательно, она делится на x-x0. Тогда разделенная разность первого порядка $P_n(x, x_0) = \frac{P_n(x) - P_n(x_0)}{x - x_0}$ - многочлен степени n-1 относительно x (и относительно x0, так как выражение симметрично относительно x и x0. Разность обращается в нуль при x=x1, поэтому, разделенная разность второго порядка $P_n(x, x_0, x_1) = \frac{P_n(x, x_0) - P_n(x_0, x_1)}{x - x_1}$ - многочлен степени n-2. Аналогично, P_n (x, x0, x1,x2)- многочлен степени n-3 и т.д.

Разделенная разность порядка n: P_n (x, x0, ..,xn)- многочлен нулевой степени. Разделенные разности более высокого порядка обращаются в нуль. Из определения разделенных разностей следует:

$$P_n(x) = P_n(x_0) + (x - x_0) \cdot P_n(x, x_0)$$

$P_n(x, x_0) = P_n(x_1, x_0) + (x - x_1) \cdot P_n(x, x_0, x_1)$ и т.д. Отсюда получаем формулу для $P_n(x)$:

$$P_n(x) = P_n(x_0) + (x - x_0) \cdot P_n(x_0, x_1) + (x - x_0) \cdot (x - x_1) \cdot P_n(x_0, x_1, x_2) + \ldots$$

$$+ (x - x_0) \cdot (x - x_1) \cdot \ldots \cdot (x - x_{n-1}) \cdot P_n(x_0, \ldots, x_n) \quad (15)$$

Разделенные разности в соответствии с рекуррентной формулой (14) выражаются через значения многочлена в узлах x0…xn. Если x0,…xn - узлы интерполяции, y0…yn- значения интерполируемой функции в этих узлах, то они однозначно определяют интерполяционный многочлен степени n, значения которого в узлах совпадают с yi. Тогда разделенные разности многочлена P_n (x) совпадают с разделенными разностями функции f(x). Поэтому интерполяционный многочлен можно записать в форме:

$$L_n(x) = f(x_0) + (x - x_0) \cdot f(x_0, x_1) + (x - x_0) \cdot (x - x_1) \cdot f(x_0, x_1, x_2) + \ldots$$

$$+ (x - x_0) \cdot (x - x_1) \cdot \ldots \cdot (x - x_{n-1}) \cdot f(x_0, \ldots, x_n) \quad (16)$$

Эта форма называется интерполяционным многочленом Ньютона с разделенными разностями.

Формула (16) более удобна для вычисления, чем запись интерполяционного многочлена в форме Лагранжа, т.к. добавление новых узлов интерполяции влечет вычисление только новых слагаемых, добавляемых к тому, что было вычислено с меньшим числом узлов. При использовании формы Лагранжа в этой ситуации требуется выполнять все вычисления заново.

Интерполяционный сплайн (кубический): Основными достоинствами сплайн-интерполяции являются её устойчивость и малая трудоемкость. Системы линейных уравнений, которые требуется решать для построения сплайнов, очень хорошо обусловлены, что позволяет получать коэффициенты полиномов с высокой точностью. В результате даже про очень больших *N* вычислительная схема не теряет устойчивость. Построение таблицы коэффициентов сплайна требует *O(N)*операций, а вычисление значения сплайна в заданной точке - всего лишь *O(log(N)).*

Пусть некоторая функция f(x) задана на отрезке [*a*, *b*], разбитом на части [x_{i-1},x_i], $a = x_0 < x_1 < \ldots < x_N = b$. Кубическим сплайном дефекта 1 называется функция *S*(*x*), которая:

- на каждом отрезке [x_{i-1}, x_i] является многочленом третьей степени;
- имеет непрерывные первую и вторую производные на всём отрезке [*a*, *b*];
- в точках x_i выполняется равенство $S(x_i) = f(x_i)$, т. е. сплайн *S*(*x*) интерполирует функцию *f* в точках x_i.

Для однозначного задания сплайна перечисленных условий недостаточно, для построения сплайна необходимо наложить какие-то дополнительные требования.

Естественным кубическим сплайном называется кубический сплайн, удовлетворяющий также граничным условиям вида: $S''(a) = S''(b) = 0$.

Теорема: Для любой функции *f* и любого разбиения отрезка [*a*,*b*] существует ровно один естественный сплайн S(x), удовлетворяющий перечисленным выше условиям.

Построение: Обозначим: $h_i = x_i - x_{i-1}$.На каждом отрезке [x_i, x_{i+1}] функция *S*(*x*) есть полином третьей степени $S_i(x)$, коэффициенты которого надо определить. Запишем для удобства $S_i(x)$ в виде:

$$S_i(x) = a_i + b_i(x - x_i) + \frac{c_i}{2}(x - x_i)^2 + \frac{d_i}{6}(x - x_i)^3$$ тогда

$$S_i(x_i) = a_i, \quad S_i'(x_i) = b_i, \quad S_i''(x_i) = c_i$$

Условия непрерывности всех производных до второго порядка включительно записываются в виде $S_i(x_{i-1}) = S_{i-1}(x_{i-1})$, $S_i'(x_{i-1}) = S_{i-1}'(x_{i-1}), S_i''(x_{i-1}) = S_{i-1}''(x_{i-1})$

а условия интерполяции в виде $S_i(x_{i-1}) = f(x_{i-1})$

Отсюда получаем формулы для вычисления коэффициентов сплайна: $a_i = f(x_i)$

$$h_i c_{i-1} + 2(h_i + h_{i+1})c_i + h_{i+1}c_{i+1} = 6\left(\frac{f_{i+1} - f_i}{h_{i+1}} - \frac{f_i - f_{i-1}}{h_i}\right)$$

$$d_i = \frac{c_i - c_{i-1}}{h_i}, \quad b_i = \frac{-1}{2}h_i c_i - \frac{1}{6}h_i^2 d_i + \frac{f_i - f_{i-1}}{h_i}$$

Если учесть, что $c_0 = c_n = 0$, то вычисление *c* можно провести с помощью метода прогонки для трёхдиагональной матрицы.

36. Методология автоматизированного проектирования. Блочно-иерархический подход к проектированию: декомпозиция, блочность, итерационность, унификация и стандартизация. Иерархические уровни проектирования. Восходящее, нисходящее, смешанное проектирование.

Проектирование, при котором все проектные решения или их часть получают путем взаимодействия человека и ЭВМ, называют **автоматизированным проектированием (ап)**, в отличие от ручного (без ЭВМ) или автоматического (без человека на промежуточных этапах). Система, реализующая ап, представляет собой *систему автоматизированного проектирования* (САПР или CAD System — Computer Aided Design System).

Основные понятия теории систем:

Система — множество элементов, находящихся в отношениях и связях между собой. **Элемент** — такая часть системы, представление о которой нецелесообразно подвергать дальнейшему членению при проектировании. ***Сложная система*** — система, характеризуемая большим числом элементов и большим числом взаимосвязей элементов. Сложность системы определяется также видом взаимосвязей элементов, свойствами целенаправленности, целостности, членимости, иерархичности, многоаспектности. **Подсистема** — часть системы (подмножество элементов и их взаимосвязей), которая имеет свойства системы. **Структура** — отображение совокупности элементов системы и их взаимосвязей; понятие структуры отличается от понятия самой системы также тем, что при описании структуры принимают во внимание лишь типы элементов и связей без конкретизации значений их параметров. **Параметр** — величина, выражающая свойство или системы, или ее части, или влияющей на систему среды. **Целенаправленность** — свойство искусственной системы, выражающее назначение системы. Это свойство необходимо для оценки эффективности вариантов системы. **Целостность** — свойство системы, характеризующее взаимосвязанность элементов и наличие зависимости выходных параметров от параметров элементов, при этом большинство выходных параметров не является простым повторением или суммой параметров элементов. **Иерархичность** — свойство сложной системы, выражающее возможность и целесообразность ее иерархического описания, т.е. представления в виде нескольких уровней, между компонентами которых имеются отношения целое-часть.

Системы ап и управления относятся к числу наиболее сложных. Их проектирование и сопровождение невозможны без системного подхода. Основной общий принцип системного подхода заключается в рассмотрении частей исследуемого явления или сложной системы с учетом их взаимодействия. Системный подход включает в себя выявление структуры системы, типизацию связей, определение атрибутов, анализ влияния внешней среды, формирование модели системы, исследование модели и возможно оптимизацию ее структуры и функционирования.

Практика показывает, что подавляющее большинство сложных систем как в природе, так и в технике имеет иерархическую внутреннюю структуру. Это связано с тем, что обычно связи элементов сложных систем различны как по типу, так и по силе, что и позволяет рассматривать эти системы как некоторую совокупность взаимозависимых подсистем. В иерархии «простое-сложное» любая функционирующая система является результатом развития более простой системы. Будучи в значительной степени отражением природных и технических систем, программные системы обычно являются иерархическими, т. е. обладают описанными выше свойствами.

На этих свойствах иерархических систем строится блочно-иерархический подход. **Блочно-иерархический подход** к проектированию использует идеи декомпозиции сложных описаний объектов и соответственно средств их создания на иерархические уровни и аспекты, вводит понятие направления проектирования (восходящее и нисходящее), устанавливает связь между параметрами соседних иерархических уровней.

Процесс разбиения сложного объекта на сравнительно независимые части получил название **декомпозиции**. При декомпозиции учитывают, что связи между отдельными частями должны быть слабее, чем связи элементов внутри частей. Кроме того, чтобы из полученных частей можно было собрать разрабатываемый объект, в процессе декомпозиции необходимо определить все виды связей частей между собой.

Характерные особенности:

1. Структуризация процесса проектирования, выражаемая **декомпозицией** проектных задач и документации, выделением стадий, этапов, *проектных процедур (*часть процесса проектирования, заканчивающаяся получением проектного решения, его оценкой или документальным представлением). Эта структуризация является сущностью блочно-иерархического подхода к проектированию.
2. Итерационный характер проектирования (процесс декомпозиции выполняется многократно, пока не получат малые блоки).
3. Стандартизация и унификация проектных решений и средств проектирования.

Стандартизация – нормотворческая деятельность, которая находит наиболее рациональные нормы, а затем закрепляет их в нормативных документах типа стандарта, инструкции, методики и

требований к разработке продукции, т.е. это комплекс средств, устанавливающих соответствие стандартам. *На этапе проектирования* с помощью стандартизации осуществляется следующее:

1) устанавливаются требования к качеству готовой продукции на основе комплексной стандартизации качественных характеристик продукции, а также сырья, деталей, комплексов, подсистем, и сопутствующих изделий с учетом интересов потребителей и изготовителей;

2) определяется единая система показателей качества продукции в зависимости от назначения изделий в условиях эксплуатации;

3) устанавливаются нормы, требования и методы проектирования продукции для обеспечения оптимального качества и исключения нерационального многообразия вида, марок и типоразмеров;

4) обеспечивается высокий уровень унификации производства, механизации и автоматизации производственных процессов.

Унификация (низшая ступень стандартизации) заключ в уменьшении многообразия объектов, выполняющих одинаковые или сходные функции. Унификация проводится путем агрегатирования, использования изделий или готовых модулей из ограниченного числа уже освоенных и испытанных.

Уровни проектирования

При использовании блочно-иерархического подхода к проектированию представления о проектируемой системе расчленяют на иерархические уровни. На верхнем уровне используют наименее детализированное представление, отражающее только самые общие черты и особенности проектируемой системы. На следующих уровнях степень подробности описания возрастает, при этом рассматривают уже отдельные блоки системы, но с учетом воздействий на каждый из них его соседей. Такой подход позволяет на каждом иерархическом уровне формулировать задачи приемлемой сложности, поддающиеся решению с помощью имеющихся средств проектирования. Список иерархических уровней в каждом приложении может быть специфичным, но для большинства приложений характерно следующее наиболее крупное выделение уровней:

- ***системный уровень***, на котором решают наиболее общие задачи проектирования систем, машин и процессов; результаты проектирования представляют в виде структурных схем, генеральных планов, схем размещения оборудования, диаграмм потоков данных и т.п.;
- ***макроуровень***, на котором проектируют отдельные устройства, узлы машин и приборов; результаты представляют в виде функциональных, принципиальных и кинематических схем, сборочных чертежей и т.п.;
- ***микроуровень***, на котором проектируют отдельные детали и элементы машин и приборов.

В зависимости от последовательности решения задач иерархических уровней различают **нисходящее, восходящее и смешанное проектирование**. Последовательность решения задач от нижних уровней к верхним характеризует **восходящее проектирование**, обратная последовательность приводит к **нисходящему проектированию**, в **смешанном** имеются элементы как восходящего, так и нисходящего проектирования. В большинстве случаев для сложных систем предпочитают нисходящее проектирование. Отметим однако, что при наличии заранее спроектированных составных блоков (устройств) можно говорить о смешанном проектировании. Неопределенность и нечеткость исходных данных при нисходящем проектировании (так как еще не спроектированы компоненты) или исходных требований при восходящем проектировании (поскольку ТЗ имеется на всю систему, а не на ее части) обусловливают необходимость прогнозирования недостающих данных с последующим их уточнением, т.е. последовательного приближения к окончательному решению (**итерационность**).

37. Классическое исчисление высказываний. Аксиомы и правила вывода. Вывод формул и вывод формул из гипотез. Теоремы о дедукции. Теоремы полноты и непротиворечивости.

Высказывание — это предложение, выражающее определённое суждение. Чаще всего, предложения записываются на естественном языке с некоторым набором математических символов. Высказывания должны быть либо истинными, либо ложными. Высказывание может содержать переменную. Переменная — это некоторое выражение на используемом языке, которая служит обозначением объекта из некоторого множества объектов. Это множество объектов является фиксированным и называется областью возможных значения этой переменной.

Логика высказываний (пропозициональная логика, исчисление высказываний ИВ) — это формальная теория, основным объектом которой служит понятие логического высказывания. Она является простейшей логикой, максимально близкой к человеческой логике неформальных рассуждений и известна ещё со времён античности. **Логическое высказывание —** утверждение, которому всегда можно поставить в соответствие одно из двух логических значений: ложь (0, ложно, false) или истина (1, истинно, true). Логическое высказывание принято обозначать заглавными латинскими буквами. **Высказывательной формой (пропозиц переменная)** наз логическое высказывание, в котором один из объектов заменён переменной. При подстановке вместо переменной какого-либо значения высказывательная форма превращается в высказывание. **Пример**: A(x) = «В городе x идет дождь.» A — высказывательная форма, x — объект.

Базовыми понятиями логики высказываний являются пропозициональная переменная — переменная, значением которой может быть логическое высказывание, — и **(пропозициональная) формула**, определяемая индуктивно следующим образом:

1. пропозициональные переменные являются формулами;
2. если p, q — формулы, то $(p \to q)$, $(p \vee q)$ и $(p \wedge q)$ — формулы.
3. если p — формула, то $(\neg p)$ — формула.

Подформулой наз часть формулы, сама являющаяся формулой. **Собственной подформулой** наз подформула, не совпадающая со всей формулой.

Кроме пропозициональных переменных, в ИВ используются так называемые ***логические связки***. Если p — высказывание, то через $\neg p$ будем обозначать отрицание этого высказывания. Значение двуместных логических связок $\to$ (импликация), $\wedge$ (конъюнкция) и $\vee$ (дизъюнкция) определяются так.

p	q	$p \to q$	$p \wedge q$	$p \vee q$
0	0	1	0	0
0	1	1	0	1
1	0	0	0	1
1	1	1	1	1

Формула является ***тавтолгией***, если она истинна при любых значениях входящих в нее переменных. Вот несколько широко известных примеров тавтологий логики высказываний:

Законы де Моргана.

1. $((p \wedge q) \vee (p \wedge r)) \leftrightarrow (p \wedge (q \vee r))$
2. $((p \vee q) \wedge (p \vee r)) \leftrightarrow (p \vee (q \wedge r))$

Закон контрапозиции. $(p \to q) \leftrightarrow (\neg q \to \neg p)$

Законы поглощения. $p \vee (p \wedge q) \leftrightarrow p$ *и* $p \wedge (p \vee q) \leftrightarrow p$

Законы дистрибутивности. $p \wedge (q \vee r) \leftrightarrow (p \wedge q) \vee (p \wedge r)$ и $p \vee (q \wedge r) \leftrightarrow (p \vee q) \wedge (p \vee r)$

Все тавтологии можно получить из некоторого набора "аксиом" с помощью "правил вывода", которые имеют чисто синтаксический характер и никак не апеллируют к смыслу формулы, ее истинности и т. д. **Аксиомы**.

A_1: $(A \to (B \to A))$ A_2: $((A \to (B \to C)) \to ((A \to B) \to (A \to C))$

A_3: $A \wedge B \to A$ A_4: $A \wedge B \to B$

A_5: $A \to (B \to (A \wedge B))$ A_6: $A \to (A \vee B)$

A_7: $B \to (A \vee B)$ A_8: $(A \to C) \to ((B \to C) \to ((A \vee B) \to C))$

A_9: $\neg A \to (A \to B)$ A_{10}: $(A \to B) \to ((A \to \neg B) \to \neg A)$

A_{11}: $\neg\neg A \to A$

Правила вывода, которые являются отношениями на множестве формул.

1. **Правило подстановки**.

Пусть U – формула, содержащая высказывательную переменную A. Тогда если U – истинная формула в исчислении высказываний, то, заменяя в ней переменную A всюду, куда она входит, произвольной формулой B, мы также получим истинную формулу.

2. **Правило заключения** $\dfrac{A \to B, A}{B}$ (*Modus ponens*) Правило модус поненс (правило отделения, правило вывода в формальной логике) означает, что из истинности формулы *A* (малая посылка) и *A -> B* (большая посылка) следует истинность *B*.

Пропозициональная формула A наз **выводимой** в исчислении высказываний, или **теоремой ИВ**, если существует вывод, в котором последняя формула равна A.

Теорема корректности (непротиворечивости) ИВ утверждает, что все перечисленные выше аксиомы являются тавтологиями, а с помощью правила modus ponens из истинных высказываний можно получить только истинные. **Теорема о полноте** ИВ утверждает, что всякая тавтология есть теорема

исчисления высказываний, т.е. выводима в ИВ. Доказательство этих теоремы тривиально и сводится к непосредственной проверке. Куда более интересен тот факт, что *все остальные* тавтологии можно получить из аксиом с помощью правила вывода - это так называемая теорема полноты логики высказываний.

Пусть Г — некоторое множество формул. Выводом из Г называется конечная последовательность формул, каждая из которых является аксиомой, принадлежит Г или получается из предыдущих по правилу MP. (Другими словами, мы как бы добавляем формулы из Г к аксиомам исчисления высказываний — именно как формулы, а не как схемы аксиом.) Формула А выводима из Г, если существует вывод из Г, в котором она является последней формулой. В этом случае мы пишем Г ⊦ А. Если пусто, то речь идет о выводимости в исчислении высказываний, и пишут просто ⊦ А.

Теорема о дедукции. 1)Пусть Г — множество формул. Тогда Г ⊦ А→В тогда и только тогда, когда Г U{A} ⊦ B .

2)Для произвольных формул P и Q. Если принять в качестве гипотез истинность или ложность формул P и Q, являющихся частями конъюнкции, дизъюнкции или импликации, то можно будет доказать или опровергнуть всю формулу

$$P, Q \vdash (P \wedge Q);$$
$$P, \neg Q \vdash \neg(P \wedge Q);$$
$$\neg P, Q \vdash \neg(P \wedge Q);$$
$$\neg P, \neg Q; \vdash \neg(P \wedge Q)$$

$$P, Q \vdash (P \vee Q);$$
$$P, \neg Q \vdash (P \vee Q);$$
$$\neg P, Q \vdash (P \vee Q);$$
$$\neg P, \neg Q \vdash \neg(P \vee Q);$$

$$P, Q \vdash (P \to Q);$$
$$P, \neg Q \vdash \neg(P \to Q);$$
$$\neg P, Q \vdash (P \to Q);$$
$$\neg P, \neg Q \vdash (P \to Q);$$

$$P \vdash \neg(\neg P);$$
$$\neg P \vdash \neg P.$$

38. Предикаты и кванторы. Предикатные формулы. Аксиомы и правила вывода. Вывод предикатных формул и вывод предикатных формул из гипотез.

Логика высказываний (пропозициональная логика, исчисление высказываний ИВ) — это формальная теория, основным объектом которой служит понятие логического высказывания. Она является простейшей логикой, максимально близкой к человеческой логике неформальных рассуждений и известна ещё со времён античности. **Логическое высказывание** — утверждение, которому всегда можно поставить в соответствие одно из двух логических значений: ложь (0, ложно, false) или истина (1, истинно, truth). Логическое высказывание принято обозначать заглавными латинскими буквами. Понятие **предикат** обобщает понятие высказывание. Неформально говоря, предикат – это высказывание, в которое можно подставлять аргументы.

. **Предика́т** (*n*-местный, или *n*-арный) — это функция с областью значений {0,1} (или «Истина» и «Ложь»), определённая на *n*-й декартовой степени множества *M*. Таким образом, каждую *n*-ку элементов *M* он характеризует либо как «истинную», либо как «ложную». *n*-ая Декартова степень множества *X* определяется для целых неотрицательных n, как *n*-кратное Декартово произведение *X* на себя. Прямое или декартово произведение множеств — множество, элементами которого являются всевозможные упорядоченные пары элементов исходных двух множеств.

Предикат называют *тождественно-истинным* и пишут: $P(x_1, \ldots, x_n) \equiv 1$, если на любом наборе аргументов он принимает значение 1. Предикат называют *тождественно-ложным* и пишут: $P(x_1, \ldots, x_n) \equiv 0$, если на любом наборе аргументов он принимает значение 0. Предикат называют *выполнимым*, если хотя бы на одном наборе аргументов он принимает значение 1. Так как предикаты принимают только два значения, то к ним применимы все операции булевой алгебры, например: отрицание, импликация, конъюнкция, дизъюнкция и т.д.

$B(x, y) = \{x^2 + y^2 = 1\}$ — *двуместный* предикат.

Квантор — общее название для логических операций, которые по предикату P(*x*) строят высказывание, характеризующее область истинности предиката P(*x*). В математической логике наиболее употребительны кванторы всеобщности $\forall$ и квантор существования $\exists$. С помощью кванторов $\forall$ (***всеобщности***) и $\exists$ (***существования***) можно строить из предикатов новые предикаты и высказывания. Для построения логики предикатов используется ***формализованный язык***, алфавит которого обычно содержит четыре группы символов:

1) предикатные переменные — выражения вида P_n^m, где *m* и *n* - неотрицательные целые числа;
2) предметные переменные x_1, x_2, x_3, K;
3) логические символы $\wedge$ (конъюнкция), $\vee$ (дизъюнкция), $\rightarrow$, $\Rightarrow$, $\supset$ (импликация), =, $\Leftrightarrow$, $\leftrightarrow$ (эквивалентность), $\neg$ (отрицание), кванторы - $\exists$, $\forall$;
4) вспомогательные символы (,) (скобки) и , (запятая).

Выражение P_n^m называется *m*-местной предикатной переменной; 0-местные предикатные переменные называются пропозициональными переменными.

Более сложные конструкции определяются индуктивно:

Терм есть символ переменной, либо имеет вид f(t1,…,tn), где f — функциональный символ арности n, а t1,…,tn — термы.

Атом имеет вид p(t1,…,tn), где p — предикатный символ арности n, а t1,…,tn — термы.

Формула — это либо атом, либо одна из следующих конструкций: $\neg F, F_1 \vee F_2, F_1 \wedge F_2, F_1 \rightarrow F_2, \forall x F, \exists x F$, где F,F1,F2 — формулы, а x — переменная. Альтернативное определение: Элементарной формулой называется всякая пропозициональная переменная, а также любое выражение вида $P(y_1, K, y_m)$, где P — какая-либо m-местная предикатная переменная ($m > 0$), а y_1, K, y_m — произвольные предметные переменные. Из элементарных формул следующим образом строятся предикатные формулы:

1) все элементарные формулы суть формулы;
2) если A и B — формулы, то выражения $(A \wedge B)$, $(A \vee B)$, $(A \Rightarrow B)$, $(A \Leftrightarrow B)$, $\neg A$ считаются формулами;
3) если A — формула, x — предметная переменная, то $\forall x\, A$ и $\exists x\, A$ есть формулы.

Например, предикатными формулами являются $P_1^0, P_0^1(x_1), \exists x_1 P_1^2(x_1, x_3)$,.

Целью логики предикатов является описание класса всех общезначимых формул. **Исчисления предикатов** – исчисление, аксиомами и выводимыми формулами которого являются предикатные формулы. Обычно классическое исчисление предикатов строится на основе того или иного варианта ***исчисления высказываний***: аксиомы классического исчисления высказываний считаются схемами аксиом исчисления предикатов, то есть любая предикатная формула, полученная из некоторой аксиомы исчисления высказываний подстановкой в нее каких-либо предикатных формул вместо пропозициональных переменных, объявляется аксиомой исчисления предикатов.

A_1: $(A \rightarrow (B \rightarrow A))$ A_2: $((A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$

A_3: $A \wedge B \to A$ A_4: $A \wedge B \to B$ A_5: $A \to (B \to (A \wedge B))$

A_6: $A \to (A \vee B)$ A_7: $B \to (A \vee B)$

A_8: $(A \to C) \to ((B \to C) \to ((A \vee B) \to C))$ A_9: $\neg A \to (A \to B)$

A_{10}: $(A \to B) \to ((A \to \neg B) \to \neg A)$ A_{11}: $\neg\neg A \to A$

Например, из аксиомы исчисления высказываний $A \to (B \to A)$, т.о. получается аксиома исчисления предикатов $(\forall x_2 P_0^1(x_2) \to (P_3^0 \to \forall x_2 P_0^1(x_2)))$. К этим аксиомам добавляются две новые схемы аксиом: A_{12}: $\forall x A \to A[t/x]$ A_{13}: $A[t/x] \to \exists x A$, где $A[t/x]$ — формула, полученная в результате подстановки терма t вместо каждой свободной переменной x, встречающейся в формуле A.

Правилами вывода исчисления предикатов являются *правило модус поненс* и следующие два правила:

- $\forall$ - правило, позволяющее из формулы $(B \Rightarrow A)$ получить формулу $(B \Rightarrow \forall x\, A)$, где A и B - произвольные предикатные формулы, причем B не содержит свободную переменную *x*;

- $\exists$ - правило, позволяющее при тех же предположениях относительно формул A, B и переменной *x* перейти от формулы $(A \Rightarrow B)$ к формуле $(\exists x\, A \Rightarrow B)$.

Вхождение переменной *y* в формулу *A* называется **связанным**, если оно есть вхождение в квантор $\forall y$ или $\exists y$, или в область действия одного из этих кванторов.

Всякое вхождение переменной *y*, не являющееся связанным, называется **свободным**. Например, в формуле $(\forall x_1 P_0^1(x_1) \& P_1^1(x_1))$ первые два вхождения переменной x_1 - связанные, а третье - свободное.

Переменная *y* называется свободной переменной формулы *A*, если она имеет свободные вхождения в *A*.

Правило модус поненс (правило отделения, правило вывода в формальной логике) означает, что из истинности формулы *A* (малая посылка) и $A \to B$ (большая посылка) следует истинность *B*. Выводом формулы *A* в исчислении предикатов называется конечная последовательность формул $A_1, \ldots, A_m$ такая, что каждая из формул A_i либо есть аксиома, либо получается из некоторых предшествующих ей формул по одному из перечисленных правил вывода, и A_m совпадает с *A*. Формула *A* выводима в исчислении предикатов, или является теоремой, если можно построить вывод этой формулы. Согласно *теореме Геделя о полноте*, всякая предикатная формула, истинная на всех моделях, выводима (по формальным правилам классич. исчисления предикатов).

39. Определение метрического пространства и их примеры. Открытые и замкнутые множества. Полные метрические пространства, их свойства. Пополнение метрических пространств.

Метрическим пространством называется картеж <*X*, *d*>, где *X* является множеством, а *d* — функцией расстояния: $X \times X \to \boldsymbol{R}$. Функция *d* должна удовлетворять следующим условиям:

1. $d(x, y) \geq 0$ (неотрицательность)
2. $d(x, y) = 0$ тогда и только тогда, когда $x = y$ (равенство)
3. $d(x, y) = d(y, x)$ (симметрия)
4. $d(x, z) \leq d(x, y) + d(y, z)$ (правило треугольника)

Примеры метрических пространств:

1. Дискретная метрика: $d(x, y) = 0$, если $x = y$, и $d(x, y) = 1$ во всех остальных случаях.
2. Вещественные числа с функцией расстояния $d(x, y) = |y - x|$
3. Евклидово пространство

Множество *O* является **открытым**, если для любой точки $x \in O$ найдётся положительное число *r*, такое, что множество точек на расстоянии меньше *r* от *x* принадлежит *O*.

Замкнутым называется множество, дополнение которого открыто.

Последовательность точек метрического пространства с метрикой *d* называется фундаментальной или последовательностью Коши, если она удовлетворяет критерию Коши: $\forall \varepsilon > 0 \; \exists N_\varepsilon : \mathrm{d}(x_n, x_m) < \varepsilon \; \forall n, m > N_\varepsilon$.

Полным называется метрическое пространство, если в нём любая фундаментальная последовательность имеет предел, лежащий в этом пространстве.

Свойства полных пространств:

1. Подпространство полного метрического пространства является полным тогда и только тогда, когда оно замкнуто.
2. Для того чтобы метрическое пространство было полным, необходимо и достаточно, чтобы в нём всякая последовательность вложенных друг в друга замкнутых шаров, радиусы которых стремятся к нулю, имели непустое пересечение.

Пополнением пространства M называют метрическое пространство M*, если:

1. *M* является подпространством *M**.
2. Замыкание *M* равно *M**.

Пример: пространство действительных чисел является пополнением пространства рациональных чисел.

40. Линейные пространства. Пространства операторов и функционалов. Нормированные и полные нормированные (банаховы) пространства. Норма линейного оператора и функционала в нормированном пространстве.

По́лем называется множество F с двумя бинарными операциями + (аддитивная операция, или сложение) и *(мультипликативная операция, или умножение), если оно (вместе с этими операциями) образует коммутативное ассоциативное кольцо с единицей 1!=0, все ненулевые элементы которого обратимы.

Пусть P — поле (например, действительные или комплексные числа), элементы которого называются скалярами. Тогда **линейное или векторное пространство** $L(P)$ над полем P — это множество L, на котором введены операции сложения двух элементов и умножения элемента множества на скаляр. Операции должны обладать следующими свойствами:

1. Ассоциативность сложения: $\forall x, y, z \in L$ верно $x + (y + z) = (x + y) + z$
2. Коммутативность сложения: $\forall x, y \in L$ верно $x + y = y + x$
3. Существует нейтральный элемент относительно сложения, то есть $\exists \theta \in L : \forall x \in L$ выполняется $\theta + x = x + \theta = x$
4. Существование противоположенного относительно сложения элемента: $\forall x \in L\ \exists y \in L : x + y = \theta$.
5. Дистрибутивность сложения относительно умножения на скаляр: $\forall a \in P, x, y \in L$ верно $a \cdot (x + y) = a \cdot x + a \cdot y$
6. Дистрибутивность умножения на скаляр относительно сложения: $\forall a, b \in P, x \in L$ верно $(a + b) \cdot x = a \cdot x + b \cdot x$
7. Ассоциативность умножения на скаляр: $\forall a, b \in P, x \in L$ верно $(a \cdot b) \cdot x = a \cdot (b \cdot x)$
8. Существование нейтрального элемента относительно умножения: $\exists \mathbf{1} \in P : \forall x \in L$ верно $\mathbf{1} \cdot x = x$

Функционалом называют числовую функцию, определённую на некотором линейном пространстве. Функционал называется **аддитивным**, если $f(x) + f(y) = f(x + y)$. Функционал называется **однородным**, если $f(\alpha \cdot x) = \alpha \cdot f(x)$. Функционал называется **линейным**, если он аддитивный и однородный. Пример линейного функционала: интеграл заданной функции на некотором интервале.

Оператором называют отображение множества функций в множество функций.

Нормированным векторным пространством называется векторное пространство, для которого введена норма. **Нормой** называется функция p: $L \to \boldsymbol{R}$, удовлетворяющая следующим условиям ($x, y \in L$, $a \in P$, где L — векторное пространство над полем P):

1. $p(x) \geq 0$
2. $p(x + y) \leq p(x) + p(y)$
3. $p(a \cdot x) = a \cdot p(x)$

Банаховым называется полное нормированное векторное пространство.

Норму линейного оператора A ($F \to E$) часто определяют как

$||A|| = sup\{ ||Ax||_F ; x \in E, ||x||_E = 1\}$

Если словами: Линейное пространство L над полем P — множество, для которого определены операция сложения двух элементов из L (ассоциативная, коммутативная, с нейтральным элементов, с обратным элементом), и операция умножения элемента поля P на элемент L (ассоциативная, дистрибутивная, с нейтральным элементом).

Линейный функционал — числовая функция f, определённую на некотором линейном пространстве, $f(x) + f(y) = f(x + y)$, $f(\alpha \cdot x) = \alpha \cdot f(x)$

41. Одномерные случайные величины. Функция распределения и ее свойства. Дискретные случайные величины и их свойства. Биномиальное распределение. Распределение Пуассона. Равномерное распределение. Нормальное распределение.

СВ - величина, принимающая в зависимости от случайного исхода испытания те или иные значения с определенными вероятностями. Так, число очков, выпадающее на верхней грани игральной кости, представляет собой случайную величину, принимающую значения 1, 2, 3, 4, 5, 6 с вероятностью 1/6 каждое.

Скалярную функцию $X(\omega)$, заданную на пространстве элементарных исходов, называют случайной величиной, если для любого $x \in R\{\omega : X(\omega) < x\}$ – множество элементарных исходов, для которых $X(\omega) < x$ является событием. Обозначения X – случайная величина, x – её возможные значения, ω – пространство элементарных исходов.

Функцией распределения (вероятностей) случайной величины X называют функцию $F(x)$, значение которой в точке x равно вероятности события $\{X < x\}$, т.е. события, состоящего из тех и только тех элементарных исходов ω, для которых $X(\omega) < x : F(x) = P\{X, x\}$. Обычно говорят, что значение функции распределения в точке *x* равно вероятности того, что случайная величина *X* примет значение, меньшее *x*.

Свойства:

1) $0 \le F(x) \le 1.$
2) $F(x_1) \le F(x_2)$ *при* $x_1 < x_2$ *(*$F(x)$ *- неубывающая функция).*
3) $F(-\infty) = \lim_{x \to -\infty} F(x) = 0;\ F(+\infty) = \lim_{x \to +\infty} F(x) = 1.$
4) $P\{x_1 \le X \le x_2\} = F(x_2) - F(x_1).$
5) $F(x) = F(x-0)$, *где* $F(x-0) = \lim_{y \to x-0} F(y)$ *(*$F(x)$ *- непрерывна справа)*

Непрерывной называют случайную величину X функцию распределения которой F(x) можно представить в виде $F(x) = \int_{-\infty}^{x} p(y)dy$. Функцию p(x) называют **плотностью распределения** случайной величины X или дифференциальным законом распределения, а функцию распределения F(x) называют интегральным законом распределения.

Дискретной (прерывной) называют случайную величину, которая принимает отдельные, изолированные возможные значения с определенными вероятностями. Число возможных значений дискретной случайной величины может быть конечным или бесконечным. У дискретной случайной величины функция распределения ступенчатая.

Рядом распределения дискретной случайной величины *X* называют таблицу из двух строк: верхняя – все возможные значения случайной величины, нижняя – вероятности того, что случайная величина примет эти значения.

Дискретная случайная величина X распределена по **биномиальному закону**, если она принимает значения 0, 1, …, n в соответствии с распределением, заданным формулой $P_n(i) = C_n^i p^i q^{n-i}$. Закон назван «биномиальным» потому, что правую часть равенства можно рассматривать как общий член разложения бинома Ньютона. (Cnk – сочетание) $C_n^k = \frac{n!}{k!\,(n-k)!}$.

1)Биномиальное распределение является распределением числа успехов *X* в *n* испытаниях по схеме Бернулли с вероятностью успеха *p* и неудачи *q = 1-p*. Схема Бернулли – последовательность испытаний, где каждое испытание считается независимым и имеет только два исхода: успех или неудача.

Пример. Монета брошена 2 раза. Написать в виде таблицы закон распределения случайной величины X — числа выпадений «герба».

Решение. Вероятность появления «герба» в каждом бросании монеты p=1/2, следовательно, вероятность непоявления «герба» q=1-1/2=1/2. При двух бросаниях монеты «герб» может появиться либо 2 раза, либо 1 раз, либо совсем не появиться. Таким образом, возможные значения X таковы: $x_1 = 2$, $x_2 = 1$, $x_3=0$. Найдем вероятности этих
возможных значений по формуле Бернулли:

$P_2(2) = C_2^2 p^2 = (1/2)^2 = 0{,}25.\ P_2(1) = C_2^1 pq = 2*(1/2)*(1/2) = 0{,}5.\ P_2(0) = C_2^0 p^2 = (1/2)^2 = 0{,}25$

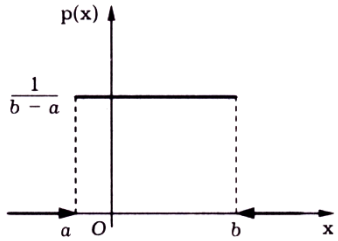
Контроль: 0,25 + 0,5+0,25=1.

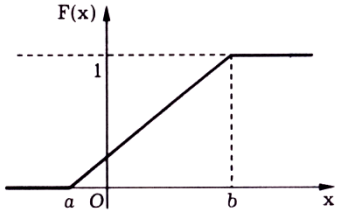
2)Дискретная случайная величина X распределена по **закону Пуассона**, если она принимает целые неотрицательные значения с вероятностями $P(i,\lambda) = \frac{\lambda^i}{i!} e^{-\lambda}$, *i = 0,1,…,* В виде ряда:

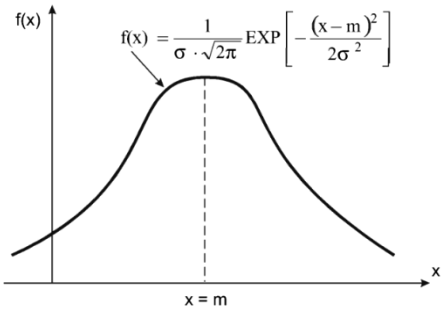
X	0	1	2	…	n	…
P	$e^{-\lambda}$	$\lambda e^{-\lambda}$	$\frac{\lambda^2}{2!}e^{-\lambda}$	…	$\frac{\lambda^n}{n!}e^{-\lambda}$	…

3)Для дискретной случайной величины **равномерное распределение** задаётся функцией распределения $F(x)=\frac{1}{N}$, для x = 1,2,…,N. Непрерывная случайная величина имеет равномерное распределение на отрезке [a,b], если её плотность распределения (p) и функция распределения (F) : $p(x)=\begin{cases}\frac{1}{b-a}, x\in[a,b]\\ 0, x\notin[a,b]\end{cases}$ и

$$F(x)\equiv P(X\le x)=\begin{cases}0, x<a\\ \frac{x-a}{b-a}, a\le x\le b\\ 1, x\ge b\end{cases}$$







4)Нормальным (гауссовым) называют распределение вероятностей непрерывной случайной величины, которое описывается плотностью

$$\varphi_{m,\sigma}(x)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-m)^2}{2\sigma^2}}\ (-\infty<m<+\infty,\sigma>0)$$

Нормальное распределение зависит от m – математического ожидания (определяет положение центра симметрии плотности распределения) и σ - среднего квадратичного отклонения (определяет разброс значений случайной величины относительно центра симметрии)

43. Числовые характеристики случайных величин: математическое ожидание и его свойства; дисперсия и ее свойства; ковариация и ее свойства.; коэффициент корреляции и его свойства. Связь между коррелированными и зависимыми случайными величинами. Ковариационная и корреляционная матрицы. Числовые характеристики основных законов распределения.

Математическое ожидание *M*(*X*) случайной величины X можно считать центром распределения этой случайной величины. Если X – дискретная случайная величина, принимающая значения *x1* , *x2* ,..., *xn* с вероятностями *p1* , *p2* ,..., *pn*, то математическое ожидание определяется по формуле $M(x) = x_1 p_1 + ... + x_n p_n = \sum_{i=1}^{n} x_i p_i$. Если X непр. с. в. и имеет плотность вероятности *f* (*x*) , тогда математическое ожидание M(X) непрерывной случайной величины X равно: $M(x) = \int_{-\infty}^{+\infty} x * f(x) dx$

Свойства: 1. Если случайная величина X принимает только одно значение C с вероятностью единица, то *M*(*C*) = *C;* 2. *M*(*aX +b*) =*aM* (*X*) +*b*, где *a*,*b – const;* 3. M(X1 + X2) = M(X1) + M(X2) 4.M(X1*X2) = M(X1)*M(X2) для независимых случайных величин

Дисперсия D(X) случайной величины X характеризует степень разброса значений этой величины около её математического ожидания. Дисперсией случайной величины X называют математическое ожидание квадрата отклонения случайной величины от её математического ожидания: $D(X) = M[X - M(X)]^2$.

Если ввести обозначение *M*(*X*) =*m*, то для дискретной случайной величины X дисперсия запишется как $D(X) = \sum_{i=1}^{n} p_i (x_i - m)^2$, а для непрерывной: $D(X) = \int_{-\infty}^{+\infty} (x-m)^2 f(x) dx$.

Свойства: 1. Если случайная величина X принимает только одно значение C с вероятностью единица, то *D*(*C*) =0; 2. *D*(*aX +b*) = $a^2 D(X)$, где *a*,*b – const ;* 3. *D*(*X*) = $M(X^2) - M^2(X)$; 4. D(X1 + X2) = D(X1) + D(X2) для независимых случайных величин; 5. D(X1 + X2) = D(X1) + D(X2) – 2cov(X1,X2)

Ковариация и ее свойства

Пусть (X1, X2) - двумерный случайный вектор. Ковариацией (корреляционным моментом) cov(X1 , X2) случайных величин *X1* , *X2* называют математическое ожидание произведения случайных величин $\dot{X}_1 = X_1 - M(X_1)$ и $\dot{X}_2 = X_2 - M(X_2)$

$\operatorname{cov}(X_1, X_2) = M(\dot{X}_1, \dot{X}_2) = M\{[X_1 - M(X_1)][X_2 - M(X_2)]\}$

Для дискретных СВ $\operatorname{cov}(X_1, X_2) = \sum_{i,j} (x_i - M(X_1))(y_j - M(X_2)) p_{ij}$

Для непрерывных СВ $\operatorname{cov}(X_1, X_2) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} (x_1 - M(X_1))(x_2 - M(X_2)) p_{X1X2} dx1 dx2$

Свойства: 1. cov(*X* , *X*) =*D*(*X*); 2. cov(X1,X2) = 0 для независимых СВ; 3. Если *Yi =ai Xi +bi* ,*i* =1,2, то cov(Y1,Y2)=a1a2 cov(X1,X2); 4. $-\sqrt{D(X1)D(X2)} \le \operatorname{cov}(X1, X2) \le \sqrt{D(X1)D(X2)}$;

5. $|\operatorname{cov}(X1, X2)| = \sqrt{D(X1)D(X2)}$, тогда и только тогда, когда *X1 u X2* связаны линейной зависимостью, т.е. существуют такие a и b, что *X2 = aX1 –b.*

6. cov(X1,X2) = M(X1,X2) – *M(X1)M(X2)*

Коэффициент корреляции и его свойства

Коэффициентом корреляции СВ-н *X u Y* называют число ρ = ρ (*X* ,*Y*) , определяемое равенством: $\rho = \frac{\operatorname{cov}(X,Y)}{\sqrt{D(X)D(Y)}}$

Свойства:

1. ρ (*X*, *X*) =1
2. Если случайные величины *X u Y* независимые, то ρ(*X*,*Y*) =0
3. ρ (a1X1+b1 , a2X2+b2) = ± ρ(X1,X2) при этом берется знак «+», если a1 и a2 имеют
4. одинаковые знаки и «-» иначе.
5. -1≤ ρ(X,Y) ≤ 1
6. ρ (*X* ,*Y*) =1,тогда и только тогда, когда *X u Y* связаны линейной зависимостью.

Связь между коррелированными и зависимыми случайными величинами

СВ-ны *X u Y* называют некоррелированными, если их ковариация cov(*X* ,*Y*) =0 и коррелированными иначе. По свойству 2) получается, что зависимые случайные величины – коррелированные.

Ковариационная и корреляционная матрицы

Матрицей ковариаций случайного вектора *X* называют матрицу Σ = (σij) = (cov(Xi,Xj)), состоящую из ковариаций СВ-н *Xi и Xj* .

Свойства:

1. Матрица ковариаций является симметрической.
2. Пусть *m-мерный* случайный вектор *Y* = (*Y1,…, Ym)* получен из *n - мерного* случайного
3. вектора *X* = (*X1,…, Xn)* с помощью линейного преобразования B, т.е. *Y= XB + c*, тогда
4. матрица ковариаций Σy случайного вектора *Y* связана с матрицей ковариации Σ*x* случайного вектора *X* соотношением $\sum_y = B^T \sum_x B$
5. Матрица ковариаций явл неотрицательно определённой, т.е. $b\Sigma b^T \geq 0$ для любого *b.*

Корреляционной матрицей случайного вектора *X* называют матрицу P = (ρij) = (ρ(Xi, Xj)) состоящую из коэффициентов корреляций случайных величин *Xi u Xj.*

Биномиальное распределение $P_n(i) = C_n^i p^i q^{n-i}$

$M(X) = np \qquad D(X_i) = pq \qquad D(X) = D(X_1) + ... + D(X_n) = npq$

Распределение Пуассона $P(i,\lambda) = \frac{\lambda^i}{i!} e^{-\lambda}$, *i = 0,1,…,*

$M(X) = \lambda \; D(X) = \lambda$

Равномерное распределение $p(x) = \begin{cases} \frac{1}{b-a}, x \in [a,b] \\ 0, x \notin [a,b] \end{cases}$

$M(X) = \frac{b+a}{2} \quad D(X) = \frac{(b-a)^2}{12}$

Нормальное распределение $\varphi_{m,\sigma}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-m)^2}{2\sigma^2}} \; (-\infty < m < +\infty, \sigma > 0)$ \

$M(X) = m \quad D(X) = \sigma^2$

44. Обработка экспериментальных данных. Сглаживание экспериментальных зависимостей. Метод наименьших квадратов. Линейная регрессионная модель. Статистический анализ регрессионной модели.

Если некоторая физическая величина зависит от другой величины, то эту зависимость можно исследовать, измеряя y при различных значениях x . В результате измерений получается ряд значений:

$x_1, x_2, \ldots, x_i, \ldots, x_n$

$y_1, y_2, \ldots, y_i, \ldots, y_n$

По данным такого эксперимента можно построить график зависимости y = f(x). Полученная кривая дает возможность судить о виде функции f(x). Однако постоянные коэффициенты, которые входят в эту функцию, остаются неизвестными. Определить их позволяет метод наименьших квадратов МНК. Экспериментальные точки, как правило, не ложатся точно на кривую. **МНК** требует, чтобы сумма квадратов отклонений экспериментальных точек от кривой, т.е. $[y_i - f(x_i)]^2$ была наименьшей. На практике этот метод наиболее часто (и наиболее просто) используется в случае линейной зависимости, т.е. когда $y = kx$ или $y = kx + b$. Рассмотрим зависимость $y = kx$. Составим величину φ – сумму квадратов отклонений наших точек от прямой $\varphi = \sum_{i=1}^{n} (y_i - kx_i)^2$. Величина φ всегда положительна и оказывается тем меньше, чем ближе к прямой лежат наши точки. Метод наименьших квадратов утверждает, что для k следует выбирать такое значение, при котором φ имеет минимум $\frac{\partial \varphi}{\partial k} = -2\sum x_i (y_i - kx_i) = 0$ или $k = \frac{\sum x_i y_i}{\sum x_i^2}$.

Рассмотрим зависимость $y = ax + b$. Если функция имеет другой вид, то необходимо провести ее линеаризацию. Например: $Z(x) = ax^b$ => LnZ = blnx + lna => z* = a*x + b*. Задача состоит в том, чтобы по имеющемуся набору значений x_i, y_i найти наилучшие значения a и b. Функция F(a,b) равна : $F(a,b) = \sum_{i=0}^{n} (ax_i + b - y_i)^2$ Для определения минимума F(a,b) вычислим ее частные производные и приравняем их к нулю:

$$F'_a = 2\sum_{i=0}^{n} (ax_i + b - y_i)x = 2\sum_{i=0}^{n} (ax_i^2 + bx_i - y_i x_i) = 0$$

После сокращения на 2 и раскрытия скобок получим систему линейных алгебраических уравнений относительно коэффициентов a и b

$$F'_b = 2\sum_{i=0}^{n} (ax_i + b - y_i) = 0$$

$$a\sum_{i=0}^{n} X_i^2 + b\sum_{i=0}^{n} x_i = \sum_{i=0}^{n} x_i y_i \quad a\sum_{i=0}^{n} X_i + b(n+1) = \sum_{i=0}^{n} y_i$$

Регрессионный анализ. Рассмотрим двумерную случайную величину(X, Y), где X и У—зависимые случайные величины. Регрессия – условное мат.ожидание переменной У, при условии, что другая переменная X принимает значение x. Тогда моделью линейной регрессии является модель, в которой мат.ожидание μ наблюдаемой случ.величины У является линейной комбинацией независимых переменных x_i, i = 1…n. Т.е. $\mu_y = \alpha 0 + \alpha_1 x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n$. Коэффициент αi – коэффициент регрессии. Для его оценки обычно используют МНК.

Функцию $g(x) = aX + b$ называют «наилучшим приближением» Y в смысле метода наименьших квадратов, если математическое ожидание $M\,[Y—g(X)]^2$ принимает наименьшее возможное значение; функцию g(x) называют среднеквадратической регрессией Y на X.

Теорема. Линейная средняя квадратическая регрессия Y на X имеет вид $g(X) = M(Y) + r\frac{\sigma_y}{\sigma_x}(X - M(X))$, где $\sigma_x = \sqrt{D(X)}, \sigma_y = \sqrt{D(Y)}, r = \frac{\mu_{xy}}{\sigma_x \sigma_y}$ - коэффициент корреляции величин X и Y. Легко убедиться, что при этих значениях a и b рассматриваемая функция принимает наименьшее значение.

50. Матрицы. Специальные виды матриц. Линейные операции над матрицами и их свойства. Транспонирование матрицы. Умножение матриц и его свойства. Прямое сложение матриц и его свойства. Определитель матрицы. Обратная матрица. Ранг матрицы. Решение матричного уравнения AX=B, где A – невырожденная матрица.

Ма́трица — математический объект, записываемый в виде прямоугольной таблицы чисел и допускающий алгебраические операции (сложение, вычитание, умножение и др.) между ним и другими подобными объектами.

Умножение матрицы на число λ заключается в построении матрицы b_{ij}, элементы которой равны произведениям соответствующих элементов исходной матрицы a_{ij} на это число $b_{ij} = \lambda a_{ij}$

Сложение матриц a_{ij} и b_{ij} заключается в нахождении третьей матрицы c_{ij}, каждый элемент которой равен сумме соответствующих элементов слагаемых. $c_{ij} = a_{ij} + b_{ij}$

Вычитание матриц определяется аналогично $c_{ij} = a_{ij} - b_{ij}$

Сложение и вычитание допускается только для матриц одинакового размера.

Существует нулевая матрица *O* такая, что её прибавление к другой матрице A не изменяет A, то есть $A + O = A$. Ясно, что все элементы нулевой матрицы должны быть равны нулю.

Умножение матриц определяется сложнее. В первом множителе должно быть столько же столбцов, сколько строк во втором. Элементы результирующей матрицы получаются как суммы произведений элементов в соответствующей строке первого множителя и столбце второго $c_{ik} = \sum a_{ij} b_{jk}$.Умножение матриц не коммутативно. Это видно хотя бы из того, что если матрицы не квадратные, то можно умножать только одну на другую, но не наоборот. Если матрицы квадратные, то результат просто меняется в зависимости от порядка сомножителей.

Пусть A- матрица размеров m x n. Тогда **транспонированной матрицей** A называется такая матрица B размеров n x m, что $b_{i,j}=a_{j,i}$, где i=1,…n, j= 1,…,m

Транспонированная матрица A обозначается A^T или A' .Операция транспонирования заключается в том, что строки и столбцы в исходной матрице меняются ролями. В транспонированной матрице первым столбцом служит первая строка исходной матрицы, вторым столбцом -- вторая строка исходной матрицы и т.д.

Для квадратных матриц существует **единичная матрица** *I* такая, что умножение любой матрицы на неё не влияет на результат, а именно $IA = AI = A$. У единичной матрицы единицы стоят только по диагонали, остальные элементы равны нулю.

Для некоторых квадратных матриц можно найти так называемую ***обратную матрицу*.** Обратная матрица A^{-1} такова, что если умножить матрицу на неё, то получится единичная матрица $AA^{-1} = I$.

Обратная матрица существует не всегда. Матрицы, для которых обратная существует, называются **невырожденными**, иначе вырожденными. Матрица невырождена, если все ее строки (столбцы) линейно независимы как векторы.

Максимальное число линейно независимых строк (столбцов) называется **рангом матрицы**.

Определитель матрицы является многочленом от элементов квадратной матрицы.

Для матрицы первого порядка **детерминантом** является сам единственный элемент этой матрицы: $\Delta = \left|a_{11}\right| = a_{11}$

Для матрицы 2×2 детерминант определяется как: $\Delta = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11}a_{22} - a_{12}a_{21}$

Для матрицы $n \times n$ определитель задаётся рекурсивно:

$\Delta = \sum_{j=1}^{n} (-1)^{1+j} a_{1j} \bar{M}_j^1$, где $\bar{M}_j^1$ — дополнительный минор к элементу a_{1j}. Эта формула называется **разложением по строке**.

В частности, формула вычисления определителя матрицы 3×3 такова:

$$\Delta = \begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11} \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} - a_{12} \begin{vmatrix} a_{21} & a_{23} \\ a_{31} & a_{33} \end{vmatrix} + a_{13} \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} =$$

$= a_{11}a_{22}a_{33} - a_{11}a_{23}a_{32} - a_{12}a_{21}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{13}a_{22}a_{31}$

Теория О. возникла в связи с задачей решения систем алгебраических уравнений 1-й степени. В наиболее важном случае, когда число уравнений равно числу неизвестных, такая система может быть записана в виде:

$$\left.\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots & \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{mn}x_n &= b_n \end{aligned}\right\}$$ или в виде AX = B

Эта система имеет одно определённое решение, если определитель $|a_{ik}|$, составленный из коэффициентов при неизвестных, не равен нулю; тогда неизвестное x_m (m = 1, 2, ..., n) равно дроби, у которой в знаменателе стоит определитель $a_{ik}|$, а в числителе - определитель, получаемый из $|a_{ik}|$ заменой элементов m-го столбца (т. е. коэффициентов при x_m) числами b_1, b_2, ..., b_n. Так, в случае системы двух уравнений с двумя неизвестными

$$\left.\begin{aligned} a_{11}x_1 + a_{12}x_2 &= b_1 \\ a_{21}x_1 + a_{22}x_2 &= b_2 \end{aligned}\right\}$$ решение даётся формулами $$x_1 = \frac{\begin{vmatrix} b_1 & a_{12} \\ b_2 & a_{22} \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}}; \quad x_2 = \frac{\begin{vmatrix} a_{11} & b_1 \\ a_{12} & b_2 \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}}.$$

Матрицы специального типа. Существует большое число различных типов М. в зависимости от выполнения различных соотношений между элементами.

Симметричная : A = A. Кососимметричная A = -A' . Ортогональная AA' = E или A^{-1}=A' . Стохастическая $a_{ij} > 0, \quad \sum_{j=1}^{n} a_{ij} = 1$.