

osn 1. Предел и непрерывность функций одной и нескольких переменных. Свойства функций непрерывных на отрезке.

osn 2. Производная и дифференциал функций одной и нескольких переменных. Достаточные условия дифференцируемости.

osn 3. Определенный интеграл, его свойства. Основная формула интегрального исчисления.

osn 4. Числовые ряды. Абсолютная и условная сходимость. Признаки сходимости: Даламбера, интегральный, Лейбница.

osn 5. Функциональные ряды. Равномерная сходимость. Признак Вейерштрасса. Непрерывность суммы равномерно сходящегося ряда непрерывных функций.

osn 6. Криволинейный интеграл, формула Грина.

osn 7. Производная функции комплексного переменного. Условия Коши-Римана. Аналитическая функция.

osn 8. Степенные ряды в действительной и комплексной области. Радиус сходимости.

osn 9. Ряд Фурье по ортогональной системе функций. Неравенство Бесселя, равнство Парсеваля, сходимость ряда Фурье.

osn 10. Прямая и плоскость, их уравнения. Взаимное расположение прямой и плоскости, основные задачи на прямую и плоскость.

osn 11. Алгебраические линии и поверхности второго порядка, канонические уравнения, классификация.

osn 12. Системы линейных алгебраических уравнений. Теорема Кронекера-Капелли. Общее решение системы линейных алгебраических уравнений.

osn 13. Линейный оператор в конечномерном пространстве, его матрица. Норма линейного оператора.

osn 14. Ортогональные преобразования евклидова пространства. Ортогональные матрицы и их свойства.

osn 15. Характеристический многочлен линейного оператора. Собственные числа и собственные векторы.

osn 16. Формализация понятия алгоритма. Машины Тьюринга, нормальные алгоритмы Маркова. Алгоритмическая неразрешимость. Задача останова. Задача самоприменимости.

osn 17. Понятие архитектуры ЭВМ. Принципы фон Неймана. Компоненты компьютера: процессор, оперативная память, внешние устройства. Аппарат прерываний.

osn 18. Операционные системы. Процессы, взаимодействие процессов, разделяемые ресурсы, синхронизация взаимодействующих процессов, взаимное исключение. Программирование взаимодействующих процессов с использованием средств ОС UNIX (сигналы, неименованные каналы, IPC).

osn 19. Системы программирования. Основные компоненты систем программирования, схема их функционирования. Общая схема работы компилятора. Основные методы, используемые при построении компиляторов.

osn 20. Основные принципы объектно-ориентированного программирования. Реализация этих принципов в языке C++. Примеры.

osn 21. Базы данных. Основные понятия реляционной модели данных. Реляционная алгебра. Средства языка запросов SQL.

osn 22. Виды параллельной обработки данных, их особенности. Компьютеры с общей распределенной памятью. Производительность вычислительных систем, методы оценки и измерения.

osn 23. Аксамбла в машинном обучении: комитеты, бэйтинг, бустинг, стекинг. Алгоритм градиентного бустинга и его параметры.

osn 24. Линейные методы в машинном обучении: линейная и гребневая регрессии, метод опорных векторов. Регуляризация в линейных методах.

osn 25. Линейные обыкновенные дифференциальные уравнения и системы. Фундаментальная система решений. Определитель Вронского.

osn 26. Теоремы существования и единственности решения задачи Коши для обыкновенного дифференциального уравнения первого порядка, разрешенного относительно производной.

osn 27. Функции алгебры логики. Реализация их формулами. Совершенная дизъюнктивная нормальная форма.

osn 28. Схемы из функциональных элементов и простейшие алгоритмы их синтеза. Оценка сложности схем, получаемых по методу Шеннона.

osn 29. Вероятностное пространство. Случайные величины. Закон больших чисел в форме Чебышева.

osn 30. Квадратурные формулы прямоугольников, трапеций и парабол.

osn 31. Методы Ньютона и секущих для решения нелинейных уравнений.

osn 32. Численное решение задачи Коши для обыкновенных дифференциальных уравнений. Примеры методов Рунге-Кutta.

osn 33. Задача Коши для уравнения колебания струны. Формула Даламбера.

osn 34. Постановка краевых задач для уравнения теплопроводности. Метод разделения переменных для решения первой краевой задачи.



Я равномерно схожусь к нежеланию ботать.

ВНИМАНИЕ!
спасибо за внимание

GOOSi
Материалы для ГОСов. Кря.

LaTeX-исходники этого материала вы можете найти здесь: <https://github.com/TheFieryLynx/GOOSi>

Мальчик, водочки нам принеси. Мы МГУ закончили.

Финальный период Второй мировой войны Гитлер провел в бункере, всячески оттягивая свой конец...

«Путин говорил, что попадет к нам», — Рай подал заявку на вступление в НАТО.

На первом свидании

Она: Мне нравятся Битлз
Они, пытаясь ее впечатлить: Офицант, принесите нам два килограмма говна, пожалуйста

— Синус, косинус, тангенс, котанганс, кунилингус — найдите лишнее слово.

— Ну... не знаю, тут четыре лишних...

Стокгольм признал безуспешными все попытки отозвать из России свой синдром.

В дверь постучали 8 раз.

— Осьминог — догадался Штирлиц

— Догадался — догадался осьминог.

Китайские астрономы обнаружили, что в России уже 22 года продолжается год Крысы.

Маленький одногодок мальчик встал не с той ноги и упал

Едут батя с сыном на шестерке, перевернулись — едут на девятке

В дверь постучали 1024 раза.

Гигабайт — подумал Штирлиц

Долбаб — ловко парировали 128 осьминогов.

Монеточка парни перед сексом:

— Выбирай, орел или решка?

Однажды в студеную зимнюю пору лошадка пипицкой примерзла к забору.

В дверь постучали 64 раза

— Восемь осьминогов, с улыбкой сказал уже подготовленный Штирлиц

— Не догадался — За дверью весело улыбались сороконожка и три осьминога.

Заходит в древнем риме мужик в бар, поднимает два пальца и говорит:

— Мне пять кружек пива, пожалуйста.

Штирлиц и Мюллер ездили по очереди на танке. Очередь редела, но не расходилась...

ООН переменился в Организацию Обеспокоенных Наций.

— А вот когда умирает черепашка, проносится ли у нее жизнь перед глазами или тиха так супермедленно проплывает?

— Я имею в виду, к свидетелю.

— К свидетелю вопросов нет, ваша честь.

Что общего между клитором и КГБ?

Одно неловкое движение языком и ты в жопе

В дверь постучали, в дверь постучали...

«Ддос-атака — хотел было подумать Штирлиц, но залагал.

Иностранный журналист спрашивает у Путина: «Господин президент, за что посадили Алексея Навального?»

— За решетку.

От работы портовой шлюхой ее останавливали только то, что в городе не было порта.

А спонсор этого дня — батюшка на батуте, вывалившись перед этим два литра пива.

Батюшка на батуте, вывалившись перед эти два литра пива — поп-рыгун

В дверь постучали 256 раз

— 32 осьминога — подумал Штирлиц

— Заебал впусти — кричал Мюллер

Песков опроверг информацию о раке у Путина, заявив, что у него краб.

Okko откроет российский аналог Pornhub «Ипокко».

В дверь вежливо постучали ногой.

— Безруков! — догадался Штирлиц.

Минкульт: в честь 9 мая будет издан ремейк знаменитой военной поэмы: «Насилуй Мародеркин».

Спрашивают у бывшей проститутки: «Как вам удалось стать миллионеркой?»

— Я всегда с собой беру ви-де-о-ка-ме-ру!!!

Встречаются два мужика в пустыне. Один говорит: «Что, гололед, да?». Второй отвечает: «Нет, с чего ты взял?». Первый ему и говорит: «А нахрена столько песка насыпали?»

Накануне голосования прокуратура ещё раз напоминает, что вмешательство граждан в выборный процесс в России недопустимо.

Аnekdot от Никитина:
Грин работал у отца на ферме, а когда отец умер — занялся математикой и спился. Можете рассказать это в б билете.

Олег перед сексом тщательно помылся, причем так тщательно, что вроде секс как уже и не нужен.

Россия объявила о победе в конкурсе «Европенавидение».

Чтобы не перепутать, бабушка назвала одного новорожденного котенка Барсик, а второго утонила.

Деду время, а потехе я посвятил жизнь

В Кремле открылся «Бункер Кинг».

А чего вы удивляетесь, что нефть стоит дешевле воды? Вы вообще нефть пробовали? Её же пить невозможно!

Мюллер выглянул в окно. По улице шел Штирлиц, ведя на поводке крохотную, зеленую с оранжевыми полосками, шестиногую собачонку.

«Странно, — подумал Мюллер, — этого анекдота я еще не знало»

По разные стороны Москвы — XXI век

дор 1. Теорема Поста о полноте систем функций в алгебре логики.
дор 2. Графы, деревья, планарные графы; их свойства. Оценка числа деревьев.
дор 3. Логика 1-го порядка. Выполнимость и общезначимость. Общая схема метода резолюции.
дор 4. Логическое программирование. Декларативная семантика и операционная семантика; соотношение между ними. Стандартная стратегия выполнения логических программ.
дор 5. Коды Бойса-Чоудхури-Хокингема: определение, алгоритмы кодирования и декодирования.
дор 6. Теорема Редфилда-Пойя (без доказательства). Примеры применения.
дор 7. Язык Python как мультипарадигмальный язык программирования. Python как интерпретатор. Объектная модель Python.
дор 8. Языки ассемблера как машиннозависимый языки низкого уровня. Организация ассемблерной программы, секции кода и данных (на примере ассемблера пасм или пасм). Основные этапы подготовки к склонке ассемблерной программы: транскляция, редактирование внешних связей (компоновка), загрузка.
дор 9. Операционные системы. Управление оперативной памятью в вычислительной системе. Алгоритмы и методы организации и управления страницей оперативной памяти.
дор 10. Зависимости в реляционных отношениях: функциональные, многозначные, проекции/соединения. Проектирование реляционных БД на основе принципов нормализации отношений. Нормальные формы.
дор 11. Закон Амдала, его следствия. Граф алгоритма. Критический путь графа алгоритма, ярусно-параллельная форма графа алгоритма. Этапы решения задачи на параллельных вычислительных системах.
дор 10. Глобальные и локальные модели освещения в компьютерной графике. Модель Фонга.
дор 11. Классификация языков, определяемых конечными автоматами, регулярными выражениями и правoliniевыми грамматиками. Эквивалентность и минимизация конечных автоматов.
дор 12. Классификация языков, определяемых конечными автоматами, регулярными выражениями и правoliniевыми грамматиками. Эквивалентность и минимизация конечных автоматов.
дор 13. Функции FIRST и FOLLOW. LL(1)-грамматики. Конструирование таблицы предсказывающего анализатора.
дор 14. Понятие имитационной модели. Примеры средств имитационного моделирования. Типовая архитектура средств имитационного моделирования (OmNet++, NS3).
дор 15. Алгоритм имитации отжига. Проблема возможности потери лучшего решения. Способы распараллеливания алгоритма имитации отжига.
дор 16. Основные понятия криптографии. Односторонняя функция с секретом. Протокол Диффи-Хеллмана выработки общего секретного ключа по открытому каналу связи.
дор 17. Основные принципы построения и архитектура сети Интернет. Алгоритмы и протоколы внешней и внутренней маршрутизации. Явление перегрузки и методы борьбы с ней.
дор 18. Теоретические основы передачи данных, физический уровень стека протоколов. Системы передачи данных Ethernet и Wi-Fi: алгоритмы работы, управление множественным доступом к каналу.
дор 19. Базисные типы данных в языках программирования. Основные проблемы, связанные с базисными типами и способы их решения в различных языках. Понятие абстрактного типа данных и способы его реализации в современных языках программирования.
дор 20. Понятие о парадигме программирования. Основные парадигмы программирования. Языки и парадигмы программирования.
дор 21. Основные характеристики функциональных языков программирования. Использование понятий функционального программирования (замыкания, анонимные функции) в современных объектно-ориентированных языках.
дор 22. Синхронизация в распределенных системах. Синхронизация времени. Логические часы. Выборы координатора. Взаимное исключение. Координация процессов.
дор 23. Отказоустойчивость в распределенных системах. Типы отказов. Фиксация контрольных точек и восстановление после отказа. Реализация и протоколы голосования. Надежная групповая рассылка.
дор 24. Распределенные файловые системы. Доступ к директориям и файлам. Семантика одновременного доступа к одному файлу нескольких процессов. Кширование: размножение файлов.
дор 25. Промежуточные представления программ: абстрактное синтаксическое дерево; последовательность трехадресных инструкций. Базовые блоки и граф потока управления.
дор 26. Локальная оптимизация при компиляции программы. Ориентированный ациклический граф и метод нумерации значений.
дор 27. Глобальная оптимизация при компиляции программы. Построение передаточных функций базовых блоков. Монотонные и дистрибутивные передаточные функции. Метод неподвижной точки и его применение для нахождения достигающих определений.
дор 28. Постановка задачи дискретной оптимизации. Метод ветвей и границ. Задача целочисленного линейного программирования.
дор 29. Комбинаторные методы нахождения оптимального пути в графе.
дор 30. Потоки в сетях. Алгоритм построения максимального потока. Оценка сложности алгоритма.

OSN 1 Предел и непрерывность функций одной и нескольких переменных. Свойства функций непрерывных на отрезке.

Множество всех упорядоченных совокупностей (x_1, \dots, x_m) из чисел x_1, \dots, x_m называется **m-мерным координатным пространством** A_m .

\exists имеется некоторое множество M и некоторая функция $\rho: M \times M \rightarrow R^+$. Функция ρ называется **метрикой** (расстоянием), а пара (M, ρ) – **метрическим пространством**, если $\forall x, y, z \in M$ выполнено:

1. $\rho(x, y) > 0$ и $\rho(x, y) = 0 \Leftrightarrow x = y$

2. $\rho(x, y) = \rho(y, x)$ (симметричность)

3. $\rho(x, y) \leq \rho(x, z) + \rho(z, y)$ (неравенство треугольника)

Если каждой точке M из $\{M\}$ точек E_m ставится в соответствие по известному закону некоторое число u , то говорят, что на множестве $\{M\}$ задана функция $u = f(M)$. $\{M\}$ – **область определения функции** $u = f(M)$. Число u , соответствующее данной M из $\{M\}$ – **значение функции** в M . Совокупность $\{u\}$ всех частных значений $u = f(M)$ – **множество значений этой функции**.

Предел по Гейне. Число $b \in R$ называется **пределенным значением** функции $u = f(M)$ в точке $A \in E_m$, если $\forall \varepsilon > 0 \exists \delta: \forall M \in \{M\}, u(M) - b| < \varepsilon$

Теорема об эквивалентности определений предела. Определение предела функции по Коши и по Гейне эквивалентны.

▲ ($\Gamma \Rightarrow K$) $\exists b$ – предел $u = f(M)$ в т. A по Гейне, но опр. по Коши не выполнено $\Rightarrow \exists \varepsilon > 0: \forall \delta > 0 \exists M \in \{M\}: 0 < \rho(M, A) < \delta, |f(M) - b| \geq \varepsilon$ $\Rightarrow \{M_n\} \rightarrow A \Rightarrow$ по Гейне $|f(M_n)| \rightarrow b \Rightarrow$ противоречие с $|f(M_n) - b| \geq \varepsilon$.

($K \Rightarrow \Gamma$) $\exists b$ – предел $u = f(M)$ в т. A по Коши и $\{M_n\} \rightarrow A$. Фиксируем $\varepsilon > 0$, по Коши $\exists \delta > 0: \forall M \in \{M\}: 0 < \rho(M, A) < \delta, |f(M) - b| < \varepsilon$. Т.к. $\{M_n\} \rightarrow A$, то для этого $\exists N \in N: \forall n \geq N, 0 < \rho(M_n, A) < \delta \Rightarrow |f(M_n) - b| < \varepsilon \Rightarrow \{f(M_n)\} \rightarrow b$

Последовательность M_1, \dots, M_n называется **фундаментальной**, если $\forall \varepsilon > 0 \exists N \in N: \forall m \geq N, p \in N$ выполнено $\rho(M_m+p, M_n) < \varepsilon$.

Критерий Коши сходимости последовательности: последовательность M_1, \dots, M_n сходится \Leftrightarrow последовательность фундаментальная.

Функция $f(M)$ удовлетворяет в точке M условию Коши, если $\forall \varepsilon > 0 \exists M', M'' \in U(M)$, удовлетворяющих $0 < \rho(M', M) < \delta, 0 < \rho(M'', M) < \delta$, следует $|f(M') - f(M'')| < \varepsilon$

Критерий Коши \exists **предела ф-ции**. Чтобы функция $f(x)$ имела конечное предельное значение в точке a , необходимо и достаточно, чтобы функция $f(a)$ удовлетворяла в этой точке условию Коши.

▲ (\Rightarrow) $\lim_{M \rightarrow A} f(M) = b$. Выберем $\varepsilon > 0 \Rightarrow$ по опр. предела по Коши для $\frac{\varepsilon}{2} > 0$, $\forall M \in \{M\}: 0 < \rho(M, A) < \delta, 0 < \rho(M'', A) < \delta \Rightarrow |f(M') - b| < \frac{\varepsilon}{2}, |f(M'') - b| < \frac{\varepsilon}{2}$. Тогда $|f(M') - f(M'')| = ||(f(M') - b) - (f(M'') - b)|| \leq |f(M') - b| + |f(M'') - b| < \frac{\varepsilon}{2} + \frac{\varepsilon}{2} < \varepsilon$

(\Leftarrow) $\exists f(M) \text{ удовл. в т. } A$. Усл. Коши: $\{M_n\} : \{M_n\} \rightarrow A, M_n \neq A$. Выберем $\varepsilon > 0$ соотв. $\delta > 0$ такое, что выше усл. Коши, для этого $\exists N \in N: \forall n \geq N \Rightarrow 0 < \rho(M_n, A) < \delta$ (т.к. $\{M_n\} \rightarrow A$). Таким образом для $p = 1, 2, \dots \Rightarrow 0 < \rho(M_{n+p}, A) < \delta$ при $n \geq N \Rightarrow$ в силу усл. Коши $|f(M_{n+p}) - f(M_n)| < \varepsilon \Rightarrow \{f(M_n)\}$ – фундаментальная $\Rightarrow \{f(M_n)\}$ сходит к некоторому b .

$\exists M_n \rightarrow A, \{M'_n\} \rightarrow A, \{f(M_n)\} \rightarrow b, \{f(M'_n)\} \rightarrow b'$. Тогда $f(M_1), f(M'_1), \dots, f(M_n), f(M'_n)$ – сходятся \Rightarrow все сёд подпосл-ти сходятся к одному пределу $\Rightarrow b = b'$.

Функция $f(x)$ называется **непрерывной** в т. a , если $\lim_{x \rightarrow a} f(x) = f(a)$ (функция должна быть задана в т. at). Для функции нескольких переменных можно определить непрерывность по каждой из переменных.

Теорема об арифметических операциях над непрерывными функциями. $\exists f(M) \text{ и } g(M)$ непрерывны в т. A . Тогда $f(M) + g(M), f(M) - g(M), f(M)g(M), f(M)/g(M)$ (последнее при условии $g(M) \neq 0$) непрерывны в т. A .

\exists функции $x_1 = \phi_1(t_1, \dots, t_k), \dots, x_m = \phi_m(t_1, \dots, t_k)$ заданы на множестве $\{N\}$ евклидовом пространстве E_m . t_1, \dots, t_k – координаты точек в E_k . $t_i \in N$ – вычисляются в соответствии с точками x_1, \dots, x_m евклидова пространства E_m . $\exists M$ – множество всех этих точек $t = (x_1, \dots, x_m)$ – функция из m переменных, заданная на $\{M\}$ – на множестве $\{N\}$ пространства E_k определена **сложная функция** $u = f(\phi_1(t_1, \dots, t_k), \dots, \phi_m(t_1, \dots, t_k)) = f(t)$

Теорема о непрерывности сложной функции. \exists функции $x_1 = \phi_1(t_1, \dots, t_k), \dots, x_m = \phi_m(t_1, \dots, t_k)$ непрерывны в т. $a = (a_1, \dots, a_k)$, а функция $u = f(x_1, \dots, x_m)$ непрерывна в т. $b = (b_1, \dots, b_m)$. Тогда сложная функция $f(t)$ непрерывна в т. a .

Свойства функций, непрерывных на отрезке (тут именно отрезок, поэтому доказываем для функции одной переменной):

Теорема о сохранении знака. $\exists f(x)$ определена на мн-ве $\{X\}$, непрерывна в т. a и $f(a) > 0$ ($f(a) < 0$). Тогда $\exists \delta > 0: \forall x \in \{X\}, x \in (a - \delta, a + \delta) \Rightarrow f(x) > 0$ ($f(x) < 0$).

▲ $\exists \varepsilon > 0 \exists \delta > 0: \forall x \in X, 0 < |x - a| < \delta \Rightarrow |f(x) - f(a)| < \varepsilon$

$\exists \varepsilon = \frac{|f(a)|}{2} \Rightarrow -\varepsilon < f(x) - f(a) < \varepsilon \Rightarrow f(a) - \frac{|f(a)|}{2} < f(x) < f(a) + \frac{|f(a)|}{2}$ (тот же знак) ■

Теорема о прохождении через 0. $\exists f(x)$ непрерывна на $[a, b]$, $f(a) > 0, f(b) < 0$. Тогда $\exists c \in (a, b): f(c) = 0$.

▲ $\exists f(a) < 0, f(b) > 0, A = \{x \in [a, b]: f(x) < 0\} \neq \emptyset$ (т.к. $a \in A$) и ограничено сверху (например, числом $b\} \Rightarrow \exists \sup A = c$. Покажем, что $f(c) = 0$.

$\exists f(c) > 0$. Тогда $c \neq a$ и по т. о сохр. знака $\exists \delta > 0: f(x) > 0 \forall x \in (c - \delta, c) \Rightarrow c \neq \sup A \Rightarrow$ противоречие $\Rightarrow f(c) \leq 0$.

$\exists f(c) < 0$. Тогда $c \neq b$ и по т. о сохр. знака $\exists \delta > 0: f(x) < 0 \forall x \in (c, c + \delta) \Rightarrow c \neq \sup A \Rightarrow$ противоречие $\Rightarrow f(c) = 0$.

Теорема о достижении значения. $\exists f(x)$ непрерывна на $[a, b]$, тогда $\forall \gamma \in [a, b]$, где $\alpha = \min\{f(a), f(b)\}, \beta = \max\{f(a), f(b)\}, \exists c \in [a, b]: f(c) = \gamma$.

▲ Если $\gamma = \alpha$ или $\gamma = \beta$ – очевидно.

$\exists \alpha < \gamma < \beta$ – очевидно.

$\exists x_1, \dots, x_n \in (a, b)$ – симметричные относительно c точки, $\{x_1, \dots, x_n\} \subset A$ и $f(x_1) = f(x_n) = \gamma$.

Теорема Больцано-Вейерштрасса (уже ниже).

Из любой ограниченной последовательности $\{x_n\}$ можно выделить сходящуюся подпоследовательность.

▲ $\exists \{X\}$ – мн-во значений последовательности $\{x_n\}$. Если $\{X\}$ – конечно, то найдется подмн-сть такая, что $x_{n_1} = x_{n_2} = x_{n_3}$. Если $\{X\}$ – бесконечно, то по принципу Больцано-Вейерштрасса (у любого отб-ка мн-ва есть хотя бы 1 предельная точка) у $\{X\}$ есть предельная точка $\Rightarrow \exists$ сходящаяся к этой точке подпосл-ть.

1-я теорема Вейерштрасса. Если $f(x)$ непрерывна на сегменте $[a, b]$, то она ограничена на нём.

▲ Выберем $\{x_n\}: x_n \in [a, b], |f(x_n)| > n$. По теореме Б-В можно выделить сход. подпосл-ть $\{x_{n_k}\}$, предел с которой в $[a, b]$. Очевидно, что посл-ть $\{f(x_{n_k})\}$ беск. большая, но в силу непр-ти функции в т. эта посл-ть должна сходиться к $f(c) \Rightarrow$ противоречие. ■

2-я теорема Вейерштрасса. Если $f(x)$ непрерывна на сегменте $[a, b]$, то она достигает на нем своих ТВГ и ТНГ.

▲ $f(x)$ непр. на $[a, b] \Rightarrow$ она огр. на $[a, b] \Rightarrow \exists M, m < M - \text{TВГ}$ и TНГ $f(x)$ на $[a, b]$. $\exists f(x) < M \forall x \in [a, b]$. Введем $g(x) = \frac{1}{M-f(x)}$.

$g(x)$ – непр. на $[a, b]$, причем знаменатель не обр. в 0 \Rightarrow огр. на $[a, b] \Rightarrow \exists A > 0: \frac{1}{M-f(x)} \leq A \forall x \in [a, b] \Rightarrow M-f(x) \geq \frac{1}{A} \Rightarrow$

$f(x) \leq M - \frac{1}{A} \forall x \in [a, b] \Rightarrow M \neq \sup f(x) \Rightarrow$ противоречие (для ТНГ аналогично) ■

Функция $f(x)$ называется **равномерно непрерывной** на множестве $\{X\}$, если для $\forall \varepsilon > 0 \exists \delta = \delta(\varepsilon) > 0: \forall x', x'' \in \{X\}: |x' - x''| < \delta$, выполняется $|f(x') - f(x'')| < \varepsilon$.

Теорема о равномерной непрерывности (Кантора). Непрерывная на сегменте $[a, b]$ функция равномерно непрерывна на нем.

▲ $\exists f(x)$ непр. на $[a, b]$, но не р/н на нем. Тогда $\exists x'_n, x''_n \in [a, b]: |x'_n - x''_n| < \frac{1}{n} \forall n \in N$, но $|f(x'_n) - f(x''_n)| \geq \varepsilon$.

$\{x'_n\} \subset \{x_n\}: \exists \lim_{n \rightarrow \infty} x'_n = c$. $\exists \{x''_n\} \subset \{x_n\}$ определению по Гейне непрерывности в точке $\{x'_n\} \rightarrow f(c), \{x''_n\} \rightarrow f(c)$ – противоречие $c \mid f(x'_n) - f(x''_n) \mid \geq \varepsilon$.

ОСN 3 Определенный интеграл, его свойства. Основная формула интегрального исчисления.

Определение:

$\exists f(x)$ задана на $[a, b], a < b, T$ – разбиение $[a, b]: a = x_0 < x_1 < \dots < x_n = b$ на n частичных сегментов $[x_0, x_1], \dots, [x_{n-1}, x_n]$. $\exists \xi_i$ – любая точка $[x_{i-1}, x_i], \Delta x_i = x_i - x_{i-1}$ – длина сегмента. $\Delta = \max(\Delta x_i)$.

Число $I\{x_i, \xi_i\}$, где $I\{x_i, \xi_i\} = f(\xi_1)\Delta x_1 + f(\xi_2)\Delta x_2 + \dots + f(\xi_n)\Delta x_n = \sum_{i=1}^n f(\xi_i)\Delta x_i$ называется **интегральной суммой** $f(x)$, соответствующей данному разбиению T сегмента $[a, b]$ и данному выбору промежуточных точек ξ_i на частичных сегментах $[x_{i-1}, x_i]$.

Число I называется **пределом интегральных сумм** $I\{x_i, \xi_i\}$ при $\delta \rightarrow 0$, если для $\forall \varepsilon > 0 \exists \delta = \delta(\varepsilon)$ для \forall разбиения T сегмента $[a, b]$, для которого $\Delta = \max(\Delta x_i) < \delta$, независимо от выбора точек ξ_i на $[x_{i-1}, x_i]$ выполняется неравенство $|I\{x_i, \xi_i\} - I| < \varepsilon$.

$$I = \lim_{\Delta \rightarrow 0} I\{x_i, \xi_i\}$$

Функция называется **интегрируемой** (по Риману) на $[a, b]$, если \exists конечный предел I интегральных сумм $f(x)$ при $\Delta \rightarrow 0$. Предел I – **определенённый интеграл** от $f(x)$ по $[a, b]: I = \int_a^b f(x)dx$

$\exists f(x)$ ограничена на $[a, b], T$ – разбиение $[a, b]$ точками $a = x_0 < x_1 < \dots < x_n = b$, M_i и m_i – точная верхняя граница и точная нижняя граница $f(x)$ на $[x_{i-1}, x_i]$. Суммы $S = \sum_{i=1}^n M_i \Delta x_i$ и $s = \sum_{i=1}^n m_i \Delta x_i$ называются **верхней и нижней суммами** $f(x)$ для данного T сегмента $[a, b]$.

$\exists \bar{T}$ – точная нижняя граница множества $\{S\}$ верхних

OSN 8 Степенные ряды в действительной и комплексной областях. Радиус сходимости.

Степенной ряд и область его сходимости.

Степенным рядом называется функциональный ряд вида

$$a_0 + \sum_{n=1}^{\infty} a_n x^n = a_0 + a_1 x + a_2 x^2 + \dots,$$

где $a_0, a_1, a_2, \dots, a_n, \dots$ — постоянные вещественные числа, называемые коэффициентами ряда.

Составим с помощью коэффициентов a_n ряда числовую последовательность:

$$\{ \sqrt[n]{|a_n|}, (n=1,2,\dots) \} \quad (1)$$

Теорема Коши-Адамара.

1. Если последовательность 1 не ограничена, то степенной ряд сходится лишь при $x = 0$.

2. Если последовательность 1 ограничена и имеет верхний предел $L > 0$, то ряд абсолютно сходится для значений x , удовлетворяющих $|x| < \frac{1}{L}$, и расходится для значений x , удовлетворяющих неравенству $|x| > \frac{1}{L}$.

3. Если последовательность 1 ограничена и ее верхний предел $L = 0$, то ряд абсолютно сходится для всех значений x .

1. Пусть последовательность 1 не ограничена. Тогда при $x \neq 0$ последовательность $|x| \sqrt[n]{|a_n|} = \sqrt[n]{|a_n x^n|}$ также не ограничена, т. е. у этой последовательности имеются члены со сколь угодно большими номерами n , удовлетворяющие неравенству $\sqrt[n]{|a_n x^n|} > 1$, или $|a_n x^n| > 1$. Но это означает, что для ряда (при $x \neq 0$) нарушено необходимое условие сходимости, т. е. ряд расходится при $x \neq 0$.

2. Пусть последовательность 1 ограничена и ее верхний предел $L > 0$. Докажем, что ряд абсолютно сходится при $|x| < \frac{1}{L}$, и расходится при $|x| > \frac{1}{L}$.

• Фиксируем начальную любое x , удовлетворяющее неравенству $|x| < \frac{1}{L}$. Тогда найдется $\varepsilon > 0$, такое, что $|x| < \frac{1}{L + \varepsilon}$.

В силу свойств верхнего предела все элементы $\sqrt[n]{|a_n|}$, начиная с некоторого номера n , удовлетворяют неравенству $\sqrt[n]{|a_n|} < L + \frac{\varepsilon}{2}$. Таким образом, начиная с этого номера n , справедливо неравенство $\sqrt[n]{|a_n x^n|} = |x| \sqrt[n]{|a_n|} < L + \frac{\varepsilon}{2} < 1$, т. е. ряд абсолютно сходится по признаку Коши.

• Фиксируем теперь любое x , удовлетворяющее неравенству $|x| > \frac{1}{L}$. Тогда найдется $\varepsilon > 0$ такое, что $|x| > \frac{1}{L - \varepsilon}$.

По определению верхнего предела из последовательности 1 можно выделить подпоследовательность $\{ \sqrt[n]{|a_n|}, (k=1,2,\dots)$, сходящуюся к L . Но это означает, что, начиная с некоторого номера k , справедливо неравенство $L - \varepsilon < \sqrt[k]{|a_n|} < L + \varepsilon$.

Таким образом, начиная с этого номера k , справедливо неравенство $\sqrt[k]{|a_n x^n|} = |x| \sqrt[k]{|a_n|} > L - \varepsilon > 1$, или $|a_n x^n| > 1$, откуда видно, что нарушено необходимое условие сходимости ряда и он расходится.

3. Пусть последовательность 1 ограничена и ее верхний предел $L = 0$. Докажем, что ряд абсолютно сходится при любом x . Фиксируем произвольное $x \neq 0$ (при $x = 0$ ряд заведомо абсолютно сходится). Поскольку верхний предел $L = 0$ и последовательность 1 не может иметь отрицательных предельных точек, число $L = 0$ является единственной предельной точкой, а следовательно, является пределом этой последовательности, т. е. последовательность является бесконечно малой. Но тогда для положительного числа $\frac{1}{2|x|}$ найдется номер, начиная с которого $\frac{1}{2|x|} < \frac{1}{2|x|}$, стало быть, начиная с указанного номера, $\sqrt[n]{|a_n x^n|} = |x|^n \sqrt[n]{|a_n|} < \frac{1}{2} < 1$, т. е. ряд абсолютно сходится к признаку Коши.

Радиус сходимости.

Теорема. Для каждого степенного ряда, если он не является рядом, сходящимся лишь в точке $x = 0$, $\exists R > 0$ (возможно, равное бесконечности) такое, что этот ряд абсолютно сходится при $|x| < R$ и расходится при $|x| > R$. Это число R называется радиусом сходимости рассматриваемого степенного ряда, а интервал $(-R, R)$ называется промежутком сходимости этого ряда. Для вычисления радиуса сходимости справедлива формула

$$R = \frac{1}{\lim_{n \rightarrow \infty} \sqrt[n]{|a_n|}}$$

(в случае, когда $\lim_{n \rightarrow \infty} \sqrt[n]{|a_n|} = 0$, $R = \infty$)

▲ Очевидно из предыдущей теоремы ■

Для случаев комплексного пространства:

Ряд вида $\sum_{n=0}^{\infty} a_n(z - z_0)^n$ называется степенным рядом с центром разложения в точке z_0 , где $\{a_n\}$ — фиксированная последовательность комплексных чисел.

Теорема Коши-Адамара.

Если $R = 0$ (т. е. $\lim_{n \rightarrow \infty} \sqrt[n]{|a_n|} = \infty$), то ряд $\sum_{n=0}^{\infty} a_n(z - z_0)^n$ сходится только в точке z_0 .

Если $R = \infty$ (т. е. $\lim_{n \rightarrow \infty} \sqrt[n]{|a_n|} = 0$), то ряд сходится абсолютно во всей комплексной плоскости C .

Если $0 < R < \infty$, то ряд сходится абсолютно внутри круга $|z - z_0| < R$, вне замкнутого круга ряд расходится.

▲ Доказательство по сути идентично доказательству для вещественного случая ■

OSN 6 Криволинейный интеграл, формула Грина.

$\square \varphi(t), \psi(t)$ непр. на $[a,b]$. Если рассматривать t как время, эти функции определяют закон движения по плоскости точки M с координатами $x = \varphi(t), y = \psi(t), \alpha < t < \beta$. Множество $\{M\}$ всех точек M , координаты x, y , которых определяются уравнениями $\varphi(t), \psi(t)$, называется простой плоской кривой L , если различным значениям параметра t из $[\alpha, \beta]$ отвечают различные точки этого множества.

$\square \varphi(t), \psi(t) \in C[t]$. Уравнения $x = \varphi(t), y = \psi(t)$ задают параметрически кривую L , если Э такая система сегментов $\{[t_{i-1}, t_i]\}$, разбивающих множество $\{t\}$, что для значений t из каждого данного сегмента этой системы все уравнения определяют простую кривую.

Справляемася кривая — кривая, имеющая конечную длину.

Длина кривой — предел последовательности длин ломаных, вспомогательных к линии, при условии, что длина наибольшего звена $\rightarrow 0$.

$\square x = \varphi(t), y = \psi(t) \in C[a, \beta]$. Тогда кривая L , определяемая x, y , спрямляемая и длину l ее дуги можно вычислить по формуле

$$l = \int_a^{\beta} \sqrt{\varphi'^2(t) + \psi'^2(t)} dt$$

\square произвольную спрямляемую кривую L на плоскости Oxy , не имеющую точек самопересечения и самонагелания, называемую ограниченной точками A, B , описываемую параметрическими уравнениями:

$$\begin{cases} x = \varphi(t) \\ y = \psi(t) \end{cases}, t \in [a, b], A = (\varphi(a), \psi(a)), B = (\varphi(b), \psi(b))$$

\square на кривой L определены три непрерывные вдоль этой кривой функции $f(x, y) = F(x, y), P(x, y), Q(x, y) = Q(M)$.

\square разбиение отрезка $[a, b] : a = t_0 < t_1 < \dots < t_n = b, \Delta t_k = t_k - t_{k-1}, M_k = M_k(\varphi(t_k), \psi(t_k))$.

Выберем точки $N_k(\xi_k, \eta_k) \in M_{k-1} M_k$, $\xi_k = \varphi(t_k), \eta_k = \psi(t_k), t_k \in [t_{k-1}, t_k]$.

\square для i интегральных сегментов:

$$1. \sigma_1 = \sum_{k=1}^n f(N_k) \Delta t_k$$

$$2. \sigma_2 = \sum_{k=1}^n P(N_k) \Delta x_k$$

$$3. \sigma_3 = \sum_{k=1}^n Q(N_k) \Delta y_k$$

Число $I_s, s = 1, 2, 3$ называется пределом интегральной суммы σ_s при $\Delta \rightarrow 0$, если $\forall \varepsilon > 0 \exists \delta > 0 : \Delta < \delta \Rightarrow |I_s - \sigma_s| < \varepsilon$ независимо от выбора точек $N_k \in M_{k-1} M_k$.

Если существует предел I_1 интегральной суммы σ_1 при $\Delta \rightarrow 0$, то он называется криволинейным интегралом 1 рода от функции f по кривой L .

• Фиксируем теперь любое x , удовлетворяющее неравенству $|x| < \frac{1}{L}$. Тогда найдется $\varepsilon > 0$ такое, что $|x| > \frac{1}{L - \varepsilon}$.

По определению верхнего предела из последовательности 1 можно выделить подпоследовательность $\{ \sqrt[n]{|a_n|}, (k=1,2,\dots)$, сходящуюся к L . Но это означает, что, начиная с некоторого номера k , спрашивается неравенство $L - \varepsilon < \sqrt[k]{|a_n|} < L + \varepsilon$.

Таким образом, начиная с этого номера k , справедливо неравенство $\sqrt[k]{|a_n x^n|} = |x|^n \sqrt[k]{|a_n|} < L - \varepsilon < 1$, или $|a_n x^n| < 1$, откуда видно, что нарушенное необходимое условие сходимости ряда и он расходится.

3. Пусть последовательность 1 ограничена и ее верхний предел $L = 0$. Докажем, что ряд абсолютно сходится при любом x . Фиксируем произвольное $x \neq 0$ (при $x = 0$ ряд заведомо абсолютно сходится). Поскольку верхний предел $L = 0$ и последовательность 1 не может иметь отрицательных предельных точек, число $L = 0$ является единственной предельной точкой, а следовательно, является пределом этой последовательности, т. е. последовательность является бесконечно малой. Но тогда для положительного числа $\frac{1}{2|x|}$ найдется номер, начиная с которого $\frac{1}{2|x|} < \frac{1}{2|x|}$, стало быть, начиная с указанного номера, $\sqrt[n]{|a_n x^n|} = |x|^n \sqrt[n]{|a_n|} < \frac{1}{2} < 1$, т. е. ряд абсолютно сходится к признаку Коши.

■ Радиус сходимости.

Теорема. Для каждого степенного ряда, если он не является рядом, сходящимся лишь в точке $x = 0$, $\exists R > 0$ (возможно, равное бесконечности) такое, что этот ряд абсолютно сходится при $|x| < R$ и расходится при $|x| > R$. Это число R называется радиусом сходимости рассматриваемого степенного ряда, а интервал $(-R, R)$ называется промежутком сходимости этого ряда. Для вычисления радиуса сходимости справедлива формула

$$R = \frac{1}{\lim_{n \rightarrow \infty} \sqrt[n]{|a_n|}}$$

(в случае, когда $\lim_{n \rightarrow \infty} \sqrt[n]{|a_n|} = 0$, $R = \infty$)

▲ Очевидно из предыдущей теоремы ■

Для случаев комплексного пространства:

Ряд вида $\sum_{n=0}^{\infty} a_n(z - z_0)^n$ называется степенным рядом с центром разложения в точке z_0 , где $\{a_n\}$ — фиксированная последовательность комплексных чисел.

Теорема Коши-Адамара.

Если $R = 0$ (т. е. $\lim_{n \rightarrow \infty} \sqrt[n]{|a_n|} = \infty$), то ряд $\sum_{n=0}^{\infty} a_n(z - z_0)^n$ сходится только в точке z_0 .

Если $R = \infty$ (т. е. $\lim_{n \rightarrow \infty} \sqrt[n]{|a_n|} = 0$), то ряд сходится абсолютно во всей комплексной плоскости C .

Если $0 < R < \infty$, то ряд сходится абсолютно внутри круга $|z - z_0| < R$, вне замкнутого круга ряд расходится.

▲ Доказательство по сути идентично доказательству для вещественного случая ■

OSN 4 Числовые ряды. Абсолютная и условная сходимость. Признаки сходимости: Даламбера, интегральный, Лейбница.

Определения.

• Рассмотрим произвольную числовую последовательность $u_1, u_2, \dots, u_k, \dots$ и формально образуем из её элементов бесконечную сумму вида $u_1 + u_2 + \dots + u_k + \dots = \sum_{k=1}^{\infty} u_k$, называемую числовым рядом.

Отдельные слагаемые u_k называются членами ряда. Сумма первых n членов ряда называется n -й частичной суммой ряда и обозначается S_n . Т. е. $S_n = u_1 + u_2 + \dots + u_n = \sum_{k=1}^n u_k$.

• Ряд называется сходящимся, если сходится последовательность $\{S_n\}$ частичных сумм этого ряда. При этом предел S указанной последовательности $\{S_n\}$ называется суммой ряда.

OSN 2 Производная и дифференциал функций одной и нескольких переменных. Достаточные условия дифференцируемости.

Производная функции $f(x)$ в точке x_0 называется предел при $\Delta x \rightarrow 0$ разностного отношения (если этот предел существует): $f'(x_0) = \lim_{\Delta x \rightarrow 0} \frac{\Delta y}{\Delta x} = \lim_{\Delta x \rightarrow 0} \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}$ ($x_0 + \Delta x \in$ области определения функции)

Функция $f(x)$ называется дифференцируемой в точке x_0 , если она определена в некоторой окрестности этой точки, а приращение Δy этой функции в точке x_0 , отвечающее приращению аргумента Δx , может быть представлено в виде $\Delta y = A \Delta x + \omega(\Delta x)$, где A — не зависящее от Δx конечное число, $\omega(\Delta x) = o(\Delta x)$ при $\Delta x \rightarrow 0$

Функция $f = f(x_1, \dots, x_m)$ называется дифференцируемой в точке $M(x_1, \dots, x_m)$, если её полное приращение в точке M можно представить:

OSN 9 Ряд Фурье по ортогональной системе функций. Неравенство Бесселя, равенство Парсеваля, сходимость ряда Фурье.

Два элемента f и g евклидова пространства называются **ортогональными**, если скалярное произведение $\langle f, g \rangle = 0$. Рассмотрим в произвольном бесконечномерном евклидовом пространстве E некоторую последовательность элементов.

$$\psi_1, \psi_2, \dots, \psi_n, \dots \quad (2)$$

Последовательность (2) называется **ортонормированной системой**, если входящие в эту последовательность элементы попарно ортогональны и имеют норму, равную единице.

Пусть в произвольном бесконечномерном евклидовом пространстве E задана произвольная ортогонализованная система элементов $\{\psi_k\}$. Рассмотрим какой угодно элемент f пространства E .

Назовём **рядом Фурье** элемента f по ортогонализованной системе $\{\psi_k\}$ ряд вид:

$$\sum_{k=1}^{\infty} f_k \psi_k,$$

в котором через f_k обозначены постоянные числа, называемые **коэффициентами Фурье** элемента f и определяемые равенствами $f_k = \langle f, \psi_k \rangle$, $k = 1, 2, \dots$

$S_n = \sum_{k=1}^n f_k \psi_k$ называется **n-й частичной суммой ряда Фурье**.

Рассмотрим наряду с n -й частичной суммой произвольную линейную комбинацию первых n элементов ортогонализованной системы $\{\psi_k\}$:

$$\sum_{k=1}^n C_k \psi_k$$

какими угодно постоянными числами C_1, C_2, \dots, C_n .

Величина $\|f - g\|$ называется **отклонением** f по норме данного евклидова пространства.

Задача о начальном приближении: $\min_{\forall \{C_i\} \in \mathbb{R}} \|f - \sum_{k=1}^n C_k \psi_k\|$

Будем минимизировать квадрат нормы:

$$\|f - \sum_{k=1}^n C_k \psi_k\|^2 = \left\langle f - \sum_{k=1}^n C_k \psi_k, f - \sum_{k=1}^n C_k \psi_k \right\rangle = \langle f, f \rangle -$$

$$2 \sum_{k=1}^n C_k \langle f, \psi_k \rangle + \sum_{k=1}^n C_k^2 = \|f\|^2 + \sum_{k=1}^n (C_k^2 - 2C_k f_k) = \left\{ \pm \sum_{k=1}^n f_k^2 \right\} =$$

$$\|f\|^2 - \sum_{k=1}^n f_k^2 + \sum_{k=1}^n (C_k - f_k)^2,$$

минимум достигается при $C_k = f_k$, $k = 1, 2, \dots, n$. Таким образом, доказана следующая теорема:

Теорема. Среди всевозможных линейных комбинаций элементов ортогонализованной системы $\{\psi_k\}$ евклидова пространства наименьшее отклонение от произвольного элемента f из пространства имеет n -ю частичную сумму ряда Фурье элемента f по системе $\{\psi_k\}$.

Следствие 1. \forall элемента f данного евклидова пространства, \forall ортогонализованной системы $\{\psi_k\}$ при произвольном выборе постоянных C_k и $\forall n$ справедливо неравенство

$$\|f\|^2 - \sum_{k=1}^n f_k^2 \leqslant \left\| \sum_{k=1}^n C_k \psi_k - f \right\|^2$$

$\Delta \|f - \sum_{k=1}^n C_k \psi_k\|^2 = \|f\|^2 - \sum_{k=1}^n f_k^2 + \sum_{k=1}^n (C_k - f_k)^2 \geq \|f\|^2 - \sum_{k=1}^n f_k^2 \blacksquare$

Следствие 2 (тождество Бесселя). \forall элемента f данного евклидова пространства, \forall ортогонализованной системы $\{\psi_k\}$ и $\forall n$ справедливо равенство

$$\left\| \sum_{k=1}^n f_k \psi_k - f \right\|^2 = \|f\|^2 - \sum_{k=1}^n f_k^2$$

Δ Подставить $C_k = f_k$ в $\|f - \sum_{k=1}^n C_k \psi_k\|^2 = \|f\|^2 - \sum_{k=1}^n f_k^2 + \sum_{k=1}^n (C_k - f_k)^2$

■

Неравенство Бесселя. \forall элемента f данного евклидова пространства, \forall ортогонализованной системы $\{\psi_k\}$ выполняется неравенство Бесселя:

$$\sum_{k=1}^n f_k^2 \leqslant \|f\|^2$$

Δ Из тождества Бесселя: $\|f - \sum_{k=1}^n f_k \psi_k\|^2 \geq 0 \Rightarrow \|f\|^2 - \sum_{k=1}^n f_k^2 \geq 0$

$0 \Rightarrow \|f\|^2 \geq \sum_{k=1}^n f_k^2 \blacksquare$

Ортогонализованная система $\{\psi_k\}$ называется **замкнутой**, если \forall элемента f данного евклидова пространства E и \forall числа $\varepsilon > 0$ найдётся такая линейная комбинация конечного числа элементов $\{\psi_k\}$, отклонение которой от f (по норме пространства E) меньше ε .

Другими словами, любой элемент пространства можно с любой степенью точности приблизить по норме этого пространства линейной комбинацией конечного числа первых элементов этой системы.

Теорема. Если ортогонализованная система $\{\psi_k\}$ является замкнутой, то \forall элемента f рассматриваемого евклидова пространства неравенство Бесселя переходит в точное равенство

$$\sum_{k=1}^{\infty} f_k^2 = \|f\|^2,$$

называемое **равенством Парсеваля**.

▲ Фиксируем произвольный элемент f рассматриваемого евклидова пространства и произвольный $\varepsilon > 0$. Т.к. система f_k является замкнутой, то найдётся такой номер n и такие числа C_1, C_2, \dots, C_n , что квадрат нормы, стоящий в правой части неравенства из следствия 1, будет меньше ε . В силу следствия 1 это означает, что для произвольного $\varepsilon > 0$ найдётся номер n , для которого $\|f\|^2 - \sum_{k=1}^n f_k^2 < \varepsilon$.

$\forall m > n$ это неравенство будет тем более справедливо, так как при возрастании номера n сумма, стоящая в левой части может только возрастать. В соединении с неравенством Бесселя это означает, что ряд сходится к сумме $\|f\|^2$. ■

[Пупкин-Залупкин, *Напиши источник!*, page 69-96]

OSN 11 Алгебраические линии и поверхности второго порядка, канонические уравнения, классификация.

Если в пространстве V_3 зафиксированы точка O и базис $\{e_1, e_2, e_3\}$, то говорят что в пространстве задана **аффинная система координат** (или **общая декартова система координат**) $\{O, e_1, e_2, e_3\}$. Точка O называется **началом координат**. Оси, проходящие через начало координат и определенные векторами $\{e_1, e_2, e_3\}$, называются **осами координат**. (Обозначается как O_{xyz}). Если вектора e_i взаимно перпендикуляры, то задана **прямоугольная система координат**.

Пусть Oxy – аффинная система координат на плоскости. **Алгебраическая линия второго порядка** определяется уравнением $F(x, y) = 0$, где $F(x, y)$ – алгебраический многочлен второй степени от переменных x и y с вещественными коэффициентами:

$$F(x, y) = a_{11}x^2 + 2a_{12}xy + a_{22}y^2 + 2a_{13}x + 2a_{23}y + a_{33} = 0, \\ a'_1 + a'_2 + a'_3 \neq 0.$$

Это ур-е называется **общим уравнением алгебраической линии второго порядка** на плоскости. Группа слагаемых $a_{11}x^2 + 2a_{12}xy + a_{22}y^2$ называется **квадратичной частью уравнения**, группа слагаемых $2a_{13}x + 2a_{23}y$ – **линейной частью**, а a_{33} – свободным членом. Введем обозначения:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{pmatrix}, \quad b = \begin{pmatrix} a_{13} \\ a_{23} \end{pmatrix}, \quad X = \begin{pmatrix} x \\ y \end{pmatrix}, \\ B = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{pmatrix} = \begin{pmatrix} A & b \\ b^T & a_{33} \end{pmatrix}$$

Тогда уравнение примет вид:

$$F(x, y) = X^T AX + 2b^T X + a_{33} = 0, \quad A \neq A^T, \quad A \neq 0.$$

Теорема. Общее уравнение линии второго порядка, заданное в прямоугольной декартовой системе координат, переходом к другой прямоугольной системе координат приводится к одному из следующих типов уравнений:

1. $\lambda_1 x^2 + \lambda_2 y^2 + a_0 = 0$, где $\lambda_1 \lambda_2 \neq 0$
2. $\lambda_2 y^2 + 2b_0 x = 0$, где $\lambda_2 b_0 \neq 0$
3. $\lambda_2 y^2 + c_0 = 0$, где $\lambda_2 \neq 0$

Эти уравнения называются **приведенными уравнениями** линии второго порядка.

▲ **Шаг 1:** (преобразование базиса). **Метод вращений**. Если $a_{12} \neq 0$, то поворотом осей можно привести квадратичную часть $F(x, y)$ к сумме квадратов: $F(x, y) = a'_{11}x'^2 + a'_{22}y'^2 + a'_{13}'x' + a'_{23}'y' + a_{33} = 0$.

Шаг 2: (перенос начала). Если в полученном ур-е содержится ненулевой квадрат какой-либо переменной, то переносом начала можно освободиться от этой переменной в первой степени. Если $a'_{11} \neq 0$ и $a'_{22} \neq 0$, то

$$x'' = x' + \frac{a'_{13}}{a'_{11}}, \quad y'' = y' + \frac{a'_{23}}{a'_{22}}, \quad a'_{33} = a_{33} - \frac{a'_{13}^2}{a'_{11}} - \frac{a'_{23}^2}{a'_{22}}$$

Все промежуточные и окончательные системы координат оставались прямоугольными, т.к. преобразования базиса с помощью ортогональной матрицы перехода сохраняют свойства ортогонализации. ■

Классификация ЛИНИЙ второго порядка

Теорема. Общее уравнение линии второго порядка, заданное в прямоугольной декартовой системе координат, определяет одну и только одну из девяти линий. Для каждой из них существует прямоугольная система координат, в которой уравнение этой линии имеет **канонический вид**:

I тип:

$$1. \frac{x^2}{a^2} + \frac{y^2}{b^2} = \pm 1 \text{ – эллипс (минимый эллипс);}$$

2. $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 0$ – пара мнимых пересекающихся прямых (пара пограничных прямых); Только начало координат удовлетворяет ур-ю мним. пер. прям.

$$3. \frac{x^2}{a^2} - \frac{y^2}{b^2} = 1 \text{ – гипербола;}$$

II тип: $y^2 = 2px$, $p > 0$ – парабола;

III тип:

1. $y^2 = \pm a^2$, $a \neq 0$ – пара параллельных прямых (пара мнимых параллельных прямых); Ни одна точка не удовлетворяет ур-ю мним. паралл. прям.

$$2. y^2 = 0$$
 – пара совпадающих прямых.

IV тип: $y^2 = 2px$, $p > 0$ – параболический цилиндр;

V тип:

1. $y^2 = \pm a^2$ – пара параллельных плоскостей (пара мнимых параллельных плоскостей);

$$2. y^2 = 0$$
 – пара совпадающих плоскостей.

VI тип: $y^2 = 2px$, $p > 0$ – параболический цилиндр;

VII тип:

1. $y^2 = \pm a^2$ – пара параллельных плоскостей (пара мнимых параллельных плоскостей);

$$2. y^2 = 0$$
 – пара совпадающих плоскостей.

■

Классификация ПОВЕРХНОСТЕЙ второго порядка

Под общим уравнением алгебраической поверхности второго порядка в системе координат Oxy пространства понимают уравнение вида:

$$F(x, y) = a_{11}x^2 + a_{22}y^2 + a_{33}z^2 + 2a_{12}xy + 2a_{13}xz + 2a_{23}yz + 2b_1x + 2b_2y + 2b_3z + c = 0$$

где a_{ij} – коэффициенты, а b_i – свободный член.

Введем обозначения:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{pmatrix}, \quad b = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}, \quad X = \begin{pmatrix} x \\ y \\ z \end{pmatrix},$$

$$B = \begin{pmatrix} a_{11} & a_{12} & a_{13} & b_1 \\ a_{12} & a_{22} & a_{23} & b_2 \\ a_{13} & a_{23} & a_{33} & b_3 \\ b_1 & b_2 & b_3 & c \end{pmatrix} = \begin{pmatrix} A & b \\ b^T & c \end{pmatrix}$$

Тогда уравнение примет вид:

OSN 16 Формализация понятия алгоритма. Машины Тьюринга, нормальные алгоритмы Маркова. Алгоритмическая неразрешимость. Задача останова. Задача самоприменимости.

Интуитивное понятие алгоритма — четкая система действий, позволяющая определенным образом обработать входные данные и выдать результат решения задачи. Важен исполнитель алгоритма. Одна и та же система действий для одного исполнителя будет алгоритмом, для другого — нет.

Алгоритм **применим** к входным данным, если исполнитель за конечное число шагов останавливается и выдаст (какой-то) ответ. В противном случае алгоритм **неприменим** в конкретных входных данных, т.е. он не останавливается, либо завершает свой выполнение аварийно (сломается).

Основные свойства алгоритма:

1. **Определенность** (понятность). Исполнитель алгоритма абсолютно точно знает, как выполнять все шаги алгоритма.

2. **Детерминированность**. Если алгоритм применим к конкретным входным данным, то он всегда и везде выдаст одинаковый ответ, а если неприменим, то всегда и везде защищается или сломается.

3. **Дискретность или структурность**. Каждый достаточно сложный шаг алгоритма тоже является алгоритмом и может быть разложен на более простые шаги. Это же касается и обрабатываемых алгоритмом данных.

Существуют разные способы формализации понятия алгоритма, Θ два из них: машины Тьюринга и нормальные алгоритмы Маркова.

Машин Тьюринга — гипотетическая машина (из-за использования бесконечной ленты). Автомат может двигаться вдоль ленты и поочереди обозревать содержимое ячеек. Он может находиться в одном из нескольких состояний q_1, \dots, q_k . В зависимости от того, какую букву s_i автомат видит в состоянии q_j , то есть от пары (s_i, q_j) (i — номер ячейки, j — номер состояния) автомат может выполнить следующие действия:

- запись новой буквы в обозреваемую ячейку;
- сдвиг влево или вправо на одну ячейку;
- переход в новое состояние.

Пример: перенести первый символ непустого слова P в его конец.

	a	b	c	Λ	комментарий
q_1	λ, R, q_2	λ, R, q_3	λ, R, q_4	λ, R	анализ I симв., удаление
q_2	λ, R	λ, R	λ, R	$a, !$	запись вправо
q_3	λ, R	λ, R	λ, R	$b, !$	запись вправо
q_4	λ, R	λ, R	λ, R	$c, !$	запись вправо

Тезис Тьюринга: если кто-то предложит какой-либо алгоритм обработки слов в заданном алфавите, то можно построить эквивалентную машину Тьюринга, которая будет применима и неприменима к однаковым множествам слов. Случай машины Тьюринга **алгоритм** — это то, что может быть реализовано МТ.

Нормальный алгоритм Маркова:

Нет понятия ленты и подразумевается непосредственный доступ к любым частям преобразуемого слова. Пусть A, B — слова в некотором алфавите. Нормальный алгоритм можно записать в следующем виде:

$A = \{ \rightarrow \} B_i$. Каждая пара — формула подстановки для замены подслов в преобразуемом слове. Ищется вхождение слова A_1 в исходное слово. Если нашли, то заменим его на B_1 , если нет, то ищем A_2 и так далее. Затем возвращаемся в начало и снова ищем вхождение A_1 . Процесс заканчивается, если ни одна из подстановок не применима, либо применялась завершающая формула, в которой \rightarrow .

Пример: $A = \{a, b\}$. Преобразовать слово P так, чтобы в его начале оказались все символы a , а в конце — все символы b .

$$\{ba \rightarrow ab\}$$

Тезис Маркова: если кто-то предложит какой-либо алгоритм обработки слов в заданном алфавите, то его можно нормализовать, т.е. построить эквивалентный нормальный алгоритм Маркова, который будет применим и неприменим к однаковым множествам слов. Машину Тьюринга и нормальные алгоритмы Маркова эквивалентны.

Самоприменимость

Входное слово, которое подаётся на вход алгоритму, может быть записью какого-то другого алгоритма. Когда алгоритм применим к своей записи, он называется **самоприменимым**.

Теорема. Если есть два алгоритма таких, что выходные данные одногожно можно использовать как входные данные для другого, то обязательно существует третий алгоритм, который работает как суперпозиция (композиция, последовательное выполнение) двух первых алгоритмов.

[Давалась без док-ва]

Задача останова

Пусть требуется построить алгоритм X , который, получая на вход запись любого алгоритма A и его конкретные входные данные D , определяет, применим ли A этим данным D (остановится ли A , получив на входе D).

Теорема. Такого алгоритма X не существует. [Давалась без док-ва]

Алгоритмическая неразрешимость

Существуют задачи, для которых в принципе невозможно построить алгоритм их решения, они и называются **алгоритмически неразрешимыми**. Пусть требуется построить алгоритм X , который, получая на вход запись любого алгоритма A , определяет, самоприменим ли этот A или нет.

Теорема. Алгоритм X не существует.

▲ Доказательство от противного. Пусть алгоритм X существует, и, получив на вход запись алгоритма A , он вырабатывает ответ DA (Да), если A самоприменим, и ответ NET (Нет), если несамоприменим. Построим вспомогательный алгоритм Y , вот его запись в форме НАМ:

$$\begin{cases} DA \rightarrow DA \\ NET \mapsto NET \end{cases}$$

Как видно, мы специально сделали так, чтобы выходные данные алгоритма X можно подать на вход алгоритма Y . Тогда обязательно существует алгоритм Z , который работает как суперпозиция $X * Y$, то есть $Z = X * Y$. Θ самоприменим ли Z .

— Пусть Z самоприменим, тогда $\langle \text{запись } Z \rangle Z \rightarrow \langle \text{запись } Z \rangle X * Y \rightarrow \langle \text{запись } Z \rangle D A \rightarrow \text{Зациклились}$, предположение неверно.

— Пусть Z несамоприменим, тогда $\langle \text{запись } Z \rangle Z \rightarrow \langle \text{запись } Z \rangle X * Y \rightarrow \langle \text{запись } Z \rangle D E N T \rightarrow \text{Стоп}$, алгоритм самоприменим, предположение неверно.

Как видно, оба предположения неверны, поэтому делаем вывод, что алгоритм Z не существует. Однако алгоритм Y существует (мы его построили), поэтому не существует алгоритм X . ■

[В. Н. Пильщиков, *Машина Тьюринга и алгоритмы Маркова. Решение задач. Учебно-методическое пособие*.]

OSN 14 Ортогональные преобразования евклидова пространства. Ортогональные матрицы и их свойства.

Базисом линейного пространства называется упорядоченная линейно независимая система векторов пространства, через которую линейно выражается любой вектор пространства.

Ортонормированный базис называется базисом, векторы которого имеют единичную длину и в случае $n > 1$ попарно перпендикуляры.

Ортогональные матрицы и их свойства.

• Матрица $Q \in \mathbb{C}^{n \times n}$ называется **унитарной**, если $Q Q^H = Q^H Q = I$

• Матрица $Q \in \mathbb{R}^{n \times n}$ называется **ортогональной**, если $Q Q^T = Q^T Q = I$

Q^H — **эрмитова-сопряженная матрица**: $q_{ij} = \bar{q}_{ji}$

Q^T — **транспонированная матрица**: $q_{ij} = q_{ji}$

Свойства ортогональной матрицы:

1. Q — обратима, причём $Q^{-1} = Q^T$;

▲ $|Q^T| = |Q| \Rightarrow |Q|^2 = 1 \Rightarrow |Q| \neq 0 \Rightarrow \exists Q^{-1}, Q^T Q = I \Rightarrow \{\text{домн. справа на } Q^{-1}\} \Rightarrow Q^T = Q^{-1}$ ■

2. $\det(Q) = \pm 1$;

▲ $|Q^T| = |Q| \Rightarrow |Q|^2 = 1 \Rightarrow |Q| = \pm 1$ ■

3. $\forall \lambda$ — собственное значение $Q \Rightarrow \lambda = \pm 1$. ▲ Пусть $Qx = \lambda x$ тогда $(x, x) = (Qx, Qx) = (Qx, x) = (\lambda x, \lambda x) = \lambda^2 (x, x) \Rightarrow (x, x) = \lambda^2 (x, x) = 1$

■

Линейный оператор U , действующий в унитарном (евклидовом) пространстве, называется **унитарным (ортогональным)** оператором, если $U^* U = U U^* = I$

• Оператор U унитарен (ортогонален) \Leftrightarrow в любом ортогональном базисе он имеет унитарную (ортогональную) матрицу.

• Для унитарного (ортогонального) оператора U справедливы равенства:

$U^* = U^{-1}, |\det U| = 1$.

• Унитарный (ортогональный) оператор нормален.

Критерий унитарности. В конечномерном унитарном (евклидовом) пространстве V следующие утверждения равносильны:

• Оператор U унитарен (ортогонален)

• $U^* U = I$

• $U U^* = I$

• U изометричен

• оператор U сохраняет длину, т.е. $|Ux| = |x|, \forall x \in V$

• оператор U переводит любой ортогональный базис V в ортогональный базис

• оператор U переводит хотя бы один ортогональный базис V в ортогональный базис

■

• $(1 \Leftrightarrow 2) \Rightarrow$ очевидно

• $(2 \Leftrightarrow 3) \Rightarrow$ аналогично

• $(3 \Leftrightarrow 4) \Rightarrow$ $(Ux, Uy) = (x, UU^*y) = (x, y)$

• $(4 \Leftrightarrow 5) \Rightarrow$ $UU^* = I \Rightarrow UU^{-1} = I \Rightarrow UU^* = I$ {домн. справа на U^{-1} }

• $(5 \Leftrightarrow 6) \Rightarrow$ очевидно

• $(6 \Leftrightarrow 7) \Rightarrow$ очевидно

• $(7 \Leftrightarrow 8) \Rightarrow$ доказано в предыдущем пункте (■)

■

Примеры ортогональных матриц.

• $Q \in \mathbb{R}^{1 \times 1} \Rightarrow Q = [\pm 1]$

• $Q \in \mathbb{R}^{2 \times 2} \Rightarrow Q = \begin{bmatrix} q_{11} & q_{12} \\ q_{21} & q_{22} \end{bmatrix}$

$QQ^T = Q^T Q = I \Rightarrow \begin{cases} q_{11}^2 + q_{21}^2 = 1 \\ q_{11}q_{12} + q_{21}q_{22} = 0 \\ q_{12}^2 + q_{22}^2 = 1 \end{cases}$

или

OSN 17 Понятие архитектуры ЭВМ. Принципы фон Неймана. Компоненты компьютера: процессор, оперативная память, внешние устройства. Аппарат прерываний.

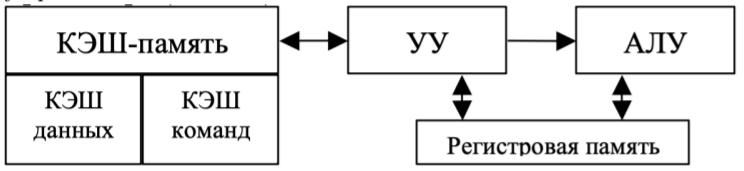
Компьютер — исполнитель алгоритма на языке машины. Архитектура ЭВМ — совокупность узлов машины и взаимосвязей между ними, рассматриваемая на определённом уровне рассмотрения этой архитектуры. Принципы фон Неймана:

1. Принцип двоичного кодирования информации: вся информация, которая поступает и обрабатывается в компьютере, кодируется в двоичной системе счисления.

2. Принцип программного управления. Программа состоит из команд, в которых закодированы операции и операнды, над которыми должна выполняться данная операция. Выполнение компьютером программы — это автоматическое выполнение определённой последовательности команд, составляющих программу. В компьютере устройство, обеспечивающее выполнение команд, — Последовательность выполненных процессором команд последовательностью команд и данных, составляющих программу. То есть, по сути, второй принцип — это принцип последовательной обработки.

3. Принцип хранения программы. Для хранения команд и данных программы используется единое устройство памяти, которое представляется в виде вектора слов. Все слова имеют последовательную адресацию. Команды и данные представляются единым образом. Интерпретация информации памяти и, соответственно, ее идентификация как команды или как данных происходит неявно при выполнении очередной команды. К примеру, содержимое слова, адрес которого используется в качестве перехода в качестве операнда, интерпретируется как команда. Если то же слово используется в качестве операнда команды сложения, то его содержимое интерпретируется как данные. То есть одна и та же область памяти в зависимости от команд в одном случае будет интерпретироваться как команда, в другом случае — как данные. Этот принцип фон Неймана замечателен тем, что он определяет возможность программной генерации команд с последующим их выполнением, то есть возможность компилиации программы, когда одна программа порождает другую программу, которая будет выполняться.

Процессор состоит из **устройств управления (УУ)** и **арифметико-логического устройства (АЛУ)**. АЛУ выполняет различные операции над данными, хранящимися на регистрах АЛУ. УУ выполняет команды языка машины, пославая управляемые сигналы к остальным устройствам.



Основная (оперативная) память хранит команды программы и обрабатываемые данные. ОЗУ состоит из ячеек, ячейка памяти — устройство, в котором размещается информация. Ячейка состоит из двух полей **тег** и **машинное слово**. Машинное слово — поле программно изменяемой информации. Здесь могут располагаться машинные команды или данные, с которыми будет оперировать программа. Имеет фиксированый для данной ЭВМ размер. Размер машинного слова — количество двоичных разрядов, размещаемых в машинном слове. Поле машинной информации (**тег**) — поле ячеек памяти, в котором схемами контроля процессорами и ОЗУ автоматически размещается информация, необходимая для осуществления контроля за целостностью и корректностью использования данных. Использование тега:

- Контроль за целостностью данных — одноразрядный тег, который использовался для контроля точности.
- Контроль доступа к командам/данным. (вся информация раскрашивается в 2 цвета - команды и данные)
- Контроль доступа к машинным типам данных. (в теге записывается код типа данных)

Ячейки памяти, расположенные не в основной памяти ЭВМ, а в других устройствах, называются регистрами. В процессе работы команды поступают на регистры в УУ, а данные — на регистры в АЛУ. АЛУ может обрабатывать данные только на своих регистрах, чтобы обработать данные, расположенные в основной памяти, их надо сначала считать на регистры в АЛУ.

Внешние устройства служат для обмена программами и данными между основной (оперативной) памятью и «внешним миром».

Аппарат прерываний — способность ЭВМ быстро и гибко реагировать на события, происходящие как внутри процессора и оперативной памяти, так и во внешних устройствах. Каждое такое событие порождает сигнал, приходящий на специальную электронную схему — контроллер прерываний.

Прерывания делятся на:

— **Внутренние (синхронные)**, источником которых являются выполняемые команды программы, их нельзя закрыть и не реагировать на них.

— **Внешние (асинхронные)**, которые вызываются событиями в периферийных устройствах. Эти прерывания можно временно закрыть, если в данный момент процессор занят другой срочной работой.

Аппаратная реакция на прерывание заключается в сохранении информации о считающейся в данное время программе (процесса) и переключение на выполнение другой программы (процедуры обработки прерываний, т.е. события, здесь включается режим блокировки прерываний). В некоторых архитектурах это называется переключением контекста. Этот механизм позволяет (при необходимости) продолжить (возобновить) выполнение прерванной программы с текущего места.

Программная реакция на прерывание производится процедурой обработчиком прерываний и делится на два этапа. Сначала происходит минимальная программная реакция, она производится в режиме с закрытыми прерываниями от внешних устройств. Это опасный режим, так как процессор не обращает внимание на все события в периферийных устройствах. Затем происходит полная программная реакция уже в режиме с открытыми прерываниями.



Короткое прерывание — обработка не требует дополнительных ресурсов ЦП и времени.

Фатальное прерывание — после него продолжить выполнение программы невозможно.

Общая схема функционирования основных компонентов СП на базе компилятора (на примере СП Си):



Общая схема функционирования основных компонентов СП на базе интерпретатора:



Пример: Выбрать среднюю зарплату продавцов, которые обслуживают покупателей их штата 'CA'.

SELECT employee.last_name, employee.salary FROM employee JOIN job using (job_id) JOIN customer ON employee_id = salesperson_id WHERE customer.state = 'CA' AND job.function = 'SALESPERSON';

[Кузнецова, Основы современных БД]

OSN 18 Системы программирования. Основные компоненты систем программирования, схема их функционирования. Общая схема работы компилятора. Основные методы, используемые при построении компиляторов.

• **Системой программирования** называется комплекс программных средств, предназначенных для поддержки программного продукта на протяжении всего жизненного цикла этого продукта.

• **Этапы жизненного цикла** программного продукта: разработка, сопровождение, эксплуатация.

• **Этапы разработки программного продукта** анализ требований, проектирование, написание текста программы ("кодирование"), трансляция, компоновка/интеграция (связывание частей программы в единую систему), верификация (процесс проверки на правильность), тестирование (обнаружение дефектов посредством сравнения с эталоном) и отладка (процесс поиска причин дефектов и их устранение), документирование, внедрение, тиражирование, сопровождение (этот этап является повторением всех предыдущих).

• **Основные компоненты системы программирования:**

1. Транслятор переводит программы на исходном языке программирования в некоторый целевой язык. В случае, когда транслятор является компилятором, целевой язык — язык ассемблера, машинный код или байт-код некоторой виртуальной машины.

2. Интерпретатор выполняет программы без необходимости предварительной компиляции в машинный код. Может содержать Just-in-Time (JIT) компилятор, который транслирует программы в машинный код во время выполнения для оптимизации часто используемых участков кода. Если интерпретатор выполняет не исходный текст, а некоторое промежуточное представление (называемое байт-кодом), то интерпретатор называется виртуальной машиной. В этом случае для получения байт-кода необходим отдельный транслятор.

3. Макрогенератор или макропроцессор выполняет преобразование текста программы, выполняя замену вызовов макроопределений их определениями. Если макропроцессор входит в состав транслятора, его называют Препроцессором. В этом случае он выполняет непосредственно трансляцию кода в целевой язык.

4. Редактор текстов используется для написания и редактирования исходного текста программ на языке программирования.

5. Редактор связей или компоновщик используется для связывания между собой (по внешним данным) объектных модулей, построенных компилятором, а также файлов библиотек (которые являются наборами объектных модулей внутри одного файла), входящих в состав СП.

6. Отладчик используется для проверочных запусков программы и исправления ошибок. В нем обычно присутствуют такие возможности как интроспекция (получение типов данных) и анализ данных программы в время выполнения, остановка выполнения в определенной точке или при определенном условии, пошаговое выполнение программы и сопоставление машинного кода программы ее исходного кода при выполнении.

7. Библиотеки стандартных программ облегчают работу программиста, используются на этапе трансляции и исполнения.

• **Дополнительные компоненты систем программирования:**

1. Система контроля версий для версионирования исходного текста ПП (git).

2. Средства конфигурирования помогают создавать различные конфигурации ПП в зависимости от конкретных параметров системного окружения.

3. Система сборки позволяет автоматизировать сборку ПО (maven)

4. Средства тестирования помогают при составлении набора тестов, автоматического выполнения тестов.

5. Профилировщик используется для анализа поведения программы и поиска критических участков кода, на которые затрачивается наибольшее количество ресурсов (пример: анализ затрачиваемого на выполнение каждой функции времени, возможно в процентах от полного времени выполнения программы). Используется для оптимизации программы.

6. Справочная система содержит справочные материалы по языку программирования и компонентам СП.

7. Инструменты для статического анализа кода позволяют найти дефекты в программном коде без выполнения программы с помощью формальных методов.

8. Средства навигации по коду позволяют более эффективно ориентироваться в коде и поддерживать, например, переход от вызова функции к ее определению.

9. Инструменты подготовки документации. (Sphinx)

10. Система управление разработкой.

В другой терминологии интерпретаторы также называют трансляторами.

В системе программирования должен обязательно присутствовать транслятор или интерпретатор, также могут присутствовать оба. В этом случае они либо взаимозаменяются (Например, tiny с компилятором может либо интерпретировать Си, либо компилировать), либо, если интерпретатор это виртуальная машина, должны использоваться одновременно (Например, Java: java+-javavm).

Схема функционирования компилятора и часто применяемые алгоритмы (методы):

1. **Лексический анализ**. Лексический анализатор читает поток символов исходного текста программы и группирует эти символы в значения последовательности — лексемы. Используется разбор с использованием регулярных грамматик для преобразования потока символов в поток лексем.

2. **Синтаксический анализ**. Синтаксический анализатор формирует из последовательности лексем (токенов) промежуточное представление программы (синтаксическое дерево, AST, дерево, parse tree). Используются алгоритмы разбора грамматик, наиболее эффективные для данной грамматики, например, рекурсивный спуск, LL(1), LR(1) или, для отдельных частей грамматики, регулярные выражения.

3. **Семантический анализ**. Семантический анализатор проверяет исходную программу на семантическую согласованность с определением языка (например, проверка типов). Используются различные обходы графов (DFS, BFS) и их комбинации.

4. Генерация промежуточного кода. Переход AST в некоторое промежуточное представление, на котором удобнее производить оптимизации. Часто применяется SSA-форма (single static assignment), в которой каждый переменной можно присвоить значение лишь единожды. Для ее построения используется обход графа для генерации промежуточного кода и алгоритм преобразования генерации ф-функций для преобразования в SSA.

5. Машинно-независимая оптимизация. Серия трансформаций промежуточного кода с целью увеличения скорости его работы на целевом процессоре с сохранением семантики работы программы. Используются различные алгоритмы работы с графиками, см. библиотеки дор25, дор26.

6. Машинно-зависимая оптимизация и генерация кода. Трансляция промежуточного представления компилятора в машинный код целевого процессора с применением наиболее эффективных для целевой машины инструкций и генерации объектного файла. Также используются различные алгоритмы работы с графиками. Для выбора инструкций — сопоставления подграфа с образцом (алгоритм пытается найти фрагменты графа, которые некоторой машинной инструкции, например для $(x < n) \& (x > (32 - n))$ может быть использован циклический сдвиг вправо) и использование знаний компилятора для выбора наиболее эффективной инструкции для данной инструкции промежуточного кода.

7. Генерация машинного кода. Создание машинного кода (байт-кода) целевого процессора на основе машинной архитектуры.

8. Сборка. Соединение всех частей машинного кода в один исполняемый файл.

9. Тестирование. Проверка исправленной программы на соответствие требований.

10. Документация. Генерация документации для пользователя.

11. Утилиты. Утилиты для поддержки разработки и эксплуатации.

12. Интеграция. Интеграция с другими системами разработки.

13. Документация. Генерация документации для пользователя.

14. Тестирование. Проверка исправленной программы на соответствие требований.

15. Документация. Генерация документации для пользователя.

16. Утилиты. Утилиты для поддержки разработки и эксплуатации.

17. Интеграция. Интеграция с другими системами разработки.

18. Документация. Генерация документации для пользователя.

19. Тестирование. Проверка исправленной программы на соответствие требований.

20. Документация. Генерация документации для пользователя.

21. Утилиты. Утилиты для поддержки разработки и эксплуатации.

22. Интеграция. Интеграция с другими системами разработки.

23. Документация. Генерация документации для пользователя.

24. Тестирование. Проверка исправленной программы на соответствие требований.

25. Документация. Генерация документации для пользователя.

26. Утилиты. Утилиты для поддержки разработки и эксплуатации.

27. Интеграция. Интеграция с другими системами разработки.

28. Документация. Генерация документации для пользователя.

29. Тестирование. Проверка исправленной программы на соответствие требований.

30. Документация. Генерация документации для пользователя.

31. Утилиты. Утилиты для поддержки разработки и эксплуатации.

32. Интеграция. Интеграция с другими системами разработки.

33. Документация. Генерация документации для пользователя.

34. Тестирование. Проверка исправленной программы на соответствие требований.

35. Документация. Генерация документации для пользователя.

36. Утилиты. Утилиты для поддержки разработки и эксплуатации.

37. Интеграция. Интеграция с другими системами разработки.

38. Документация. Генерация документации для пользователя.

39. Тестирование. Проверка исправленной программы на соответствие требований.

40. Документация. Генерация документации для пользователя.

41. Утилиты. Утилиты для поддержки разработки и эксплуатации.

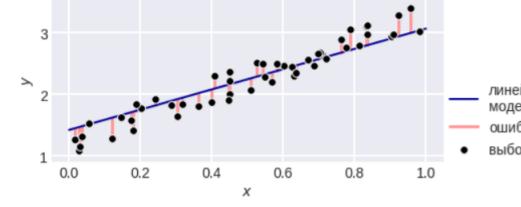
42. Интеграция. Интеграция с другими системами разработки.

OSN 24 Линейные методы в машинном обучении: линейная и гребневая регрессии, метод опорных векторов. Регуляризация в линейных методах.

Линейная регрессия

Метод определения линейных значений по формуле $a(X_1, \dots, X_n) = w_0 + w_1 X_1 + \dots + w_n X_n$ называется «линейной регрессией». Здесь X_1, \dots, X_n - значения признаков объекта.

Одномерный случай $a(X_1) = w_0 + w_1 X_1$:



Предполагая что целевые значения задаются линейно представим следующую систему:

$$\begin{cases} w_0 + w_1 x_1 = y_1, \\ \dots \\ w_0 + w_1 x_m = y_m. \end{cases}$$

Но вряд ли система решается точно (она может быть несовместна, особенно при довольно большом количестве обучающей выборки m). Формулы для «невязки» (отклонения/ошибок):

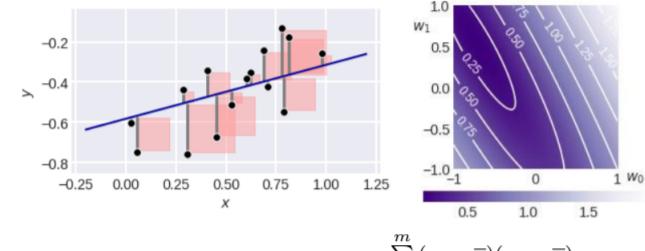
$$\begin{cases} e_1 = y_1 - w_0 - w_1 x_1, \\ \dots \\ e_m = y_m - w_0 - w_1 x_m. \end{cases}$$

Задача обучения линейной регрессии - задача минимизации суммы квадратов отклонений: $e_1^2 + \dots + e_m^2 \rightarrow \min$

Или же задача минимизации эмпирического риска по параметрам $w = (w_0, w_1)$:

$$L(w) = \sum_{i=1}^m (y_i - a_w(x_i))^2 = \sum_{i=1}^m (y_i - w_0 - w_1 x_i)^2 \rightarrow \min$$

Геометрический смысл - квадраты невязок соответствуют площадям нарисованных зелёных квадратов.



$$\text{Явное решение данной задачи: } w_1 = \frac{\sum_{i=1}^m (y_i - \bar{y})(x_i - \bar{x})}{\sum_{i=1}^m (x_i - \bar{x})^2} = \frac{\text{cov}(x_i, y_i)}{\text{var}(x_i)}$$

$$w_0 = \bar{y} - w_1 \bar{x}, \text{ где } \bar{x} = \frac{1}{m} \sum_{i=1}^m x_i \text{ и } \bar{y} = \frac{1}{m} \sum_{i=1}^m y_i$$

Общий случай $a(X_1, \dots, X_n) = w_0 + w_1 X_1 + \dots + w_n X_n$:

Здесь $w = (w_0, w_1, \dots, w_n)^T$ - вектор параметров (весов) линейной модели, $x = (X_0, X_1, \dots, X_n)^T$ - признаковое описание объекта. $X_0 \equiv 1$ - фиктивный признак.

Система уравнений:

$$\begin{cases} x_1^T w = y_1, \\ \dots \\ x_m^T w = y_m \end{cases}$$

Или же $Xw = y$.

Задача оптимизации выглядит так $\|Xw - y\|_2^2 = \sum_{i=1}^m (x_i^T w - y_i)^2 \rightarrow \min$

$$\|Xw - y\|_2^2 = (Xw - y)^T (Xw - y) = w^T X^T X w - w^T X^T y - y^T X w + y^T y \quad (7)$$

$$\nabla \|Xw - y\|_2^2 = 2X^T X w - 2X^T y = 0 \quad (8)$$

$$X^T X w = X^T y \quad (9)$$

$$w = (X^T X)^{-1} X^T y \quad (10)$$

Решение существует, если столбцы матрицы X линейно независимы.

Матрица $(X^T X)^{-1} X^T$ называется псевдообратной матрицией Мура-Пенроуза.

Правило для запоминания: исходное матричное уравнение умножить на X^T (слева и справа)

Обобщённая линейная регрессия $a(X_1, \dots, X_n) = w_0 + w_1 \phi_1(X_1, \dots, X_n) + \dots + w_k \phi_k(X_1, \dots, X_n)$: ϕ_1, \dots, ϕ_k - фиксированные базисные функции, не зависят от данных.

$$a(x) = \sum_{i=1}^k w_i \phi_i(x) = \phi(X)^T w - \text{решение.}$$

$\|\phi(X)^T w - y\|_2^2 \rightarrow \min_w$ - задача оптимизации.

Регуляризация

В $(X^T X)^{-1} X^T$ происходит обращение матрицы которая может оказаться вырожденной. Эту проблему решает регуляризация.

Упрощённое объяснение её смысла в линейной модели $a(X_1, \dots, X_n) = w_0 + w_1 X_1 + \dots + w_n X_n$: Если есть два похожих объекта, то должны быть похожи и их метки. Пусть они отличаются в j -м признаке, тогда эти модели отличаются на $\epsilon_j w_j$. Поэтому не должно быть аномально больших по модулю весов у признаков, по которым могут отличаться похожие объекты, а значит и у всех признаков X_1, \dots, X_n , поскольку заранее не известно, на каких объектах модель будет работать. Заметим также, что константного признака это рассуждение не касается.

Поэтому вместе с задачей $\|Xw - y\|_2^2 \rightarrow \min$ обычно стараются решить задачу $\|w\|_2^2 = w_1^2 + \dots + w_n^2 \rightarrow \min$ (минимизация нормы весов).

Регуляризация Иванова:

$$\begin{cases} \|Xw - y\|_2^2 \rightarrow \min, \\ \|w\|_2^2 \leq \lambda \end{cases}$$

Регуляризация Тихонова (такой вид регуляризации называется также L2-регуляризацией):

$$\begin{cases} \|Xw - y\|_2^2 + \lambda \|w\|_2^2 \rightarrow \min, \\ 0 \leq \lambda \end{cases}$$

Они эквивалентны. Решение указанной задачи регуляризации Тихонова задаётся формулой:

$$\text{argmin}(\|Xw - y\|_2^2 + \lambda \|w\|_2^2) = (X^T X + \lambda I)^{-1} X^T y \quad (11)$$

Для доказательства достаточно взять градиент и приравнять к нулю.

L1-регуляризация:

$$\begin{cases} \sum_{i=1}^m (x_i^T w - y_i)^2 + \lambda \sum_{j=1}^n |w_j| \rightarrow \min, \\ 0 \leq \lambda \end{cases}$$

Гребневая регрессия

Регрессия с коэффициентами, определяемыми формулой 11 называется гребневой регрессией (Ridge Regression). Смысл гребневой регрессии - борьба с вырожденностью (плохой обусловленностью) матрицы $X^T X$. Коэффициент λ называется коэффициентом регуляризации. При $\lambda = 0$ - обратическое решение, при $\lambda \rightarrow \infty$ матрица которую приходится обращать становиться заведомо хорошо обусловленной, метод меньше «затягивается на данные».

Отметим, что для ridge-регрессии нужна правильная нормировка признаков (как правило, стандартизация), при масштабировании (умножении признаков на скаляры) результат может отличаться.

Градиентный метод обучения

На практике не применяется аналитическое решение. Для оптимизации $\frac{1}{2} \sum_{i=1}^m (a(x_i|w) - y_i)^2 \rightarrow \min$, где $a(x|w) = w^T x$ используют итерационный метод (стохастический градиентный спуск):

$$w^{(t+1)} = w^{(t)} - n \nabla L_i(w^{(t)}) \quad (12)$$

$$w^{(t+1)} = w^{(t)} - n(a(x_i|w^{(t)}) - y_i)x_i \quad (13)$$

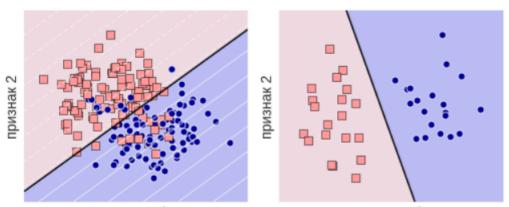
Здесь n - размер шага (скорость обучения).

Линейные классификаторы

Пусть $X = R^n$ и $Y = \{-1, 1\}$, рассмотрим линейную модель классификатора:

$$a(x) = \text{sgn}(w^T x + b) = \begin{cases} +1 & , w^T x + b \geq 0 \\ -1 & , w^T x + b < 0 \end{cases}$$

Геометрический смысл линейного классификатора - пространство делится гиперплоскостью на два полупространства:



При обучении минимизируем число ошибок:

$$L(X_{train}, a) = \sum_{t=1}^m L(y_t, a(x_t)) \rightarrow \min \quad (14)$$

$$L(y_t, a(x_t)) = \begin{cases} 1 & , \text{sgn}(w^T x_t) = y'_t, y' - \text{элемент отображения } Y = \{-1 \\ 0 & , \text{sgn}(w^T x_t) \neq y'_t \end{cases}$$

Метод опорных векторов

Линейные методы в машинном обучении: линейная и гребневая регрессии, метод опорных векторов. Регуляризация в линейных методах.

OSN 22 Виды параллельной обработки данных, их особенности. Компьютеры с общей и распределенной памятью. Производительность вычислительных систем, методы оценки и измерения.

Параллельная обработка данных имеет две разновидности: конвейерность и параллельность.

• **Параллельная обработка.** Увеличение количества независимо работающих устройств. Если некое устройство выполняет одну операцию за единицу времени, то тысячи операций оно выполнит за тысячу единиц. Система из N устройств ту же работу выполнит за $1000/N$ единиц времени (в идеальном случае).

• **Конвейерная обработка.** Усложнение самого устройства, чтобы на разных этапах могли находиться разные данные. Идея заключается в выделении отдельных этапов выполнения общей операции, причем каждый этап выполняет свою работу, передавая ее результат следующему, одновременно принимая новую порцию входных данных. Получая выигрыш в скорости обработки за счет совмещения прежде разнесенных во времени операций. Существует некоторая задержка (время разгона), для того, чтобы занять все эти этапы конвейера; когда он заполнен, происходит ускорение обработки.

Классификация многопроцессорных сетей по Флинну.

В контексте машины можно выделить два потока информации: поток управления (для передачи управляющих воздействий на конкретное устройство) и поток данных (циклический между оперативной памятью и внешними устройствами). В связи с этим выделяют 4 основных класса: SIMD (1 поток команд, 1 поток данных). "Традиционный" последовательный компьютер, SIMD (1 поток команд, много потоков данных, пример - векторные компьютеры), MIMD (много п. ком., 1. п. данных), MIMD (много и потоков ком., и данных). Среди MIMD можно выделить системы с общей ОП и системы с распределенной памятью.

• **Компьютеры с общей памятью.**

В системе присутствует несколько равноправных процессоров, имеющих одинаковый доступ к единой памяти. Всё, кроме процессоров, в одном экземпляре: образ операционной системы, память подсистема ввода-вывода и т.д. Все процессоры работают с единным адресным пространством. (+) относительная простота параллельного программирования; (-) сложность увеличения числа процессоров (рост производительности).

UMA - системы с однородным доступом к памяти (все процессоры имеют одинаковый доступ к памяти). SMP - есть общая шина, соединяющая все процессоры и ОП.

NUMA (Non Uniform Memory Access) - память физически распределена, но логически общедоступна. Каждый вычислительный узел компьютера содержит процессор, локальную память, контроллер памяти, может некоторое устройство ввода/вывода. Контроллер памяти определяет, является ли запрос к памяти локальным или его необходимо передать удаленному узлу через коммутатор/шину. Проблема - синхронизация каша.

ccNUMA (cache coherent NUMA). На аппаратном уровне решает проблему когерентности кашей. Но остаются ограничения, связанные с централизацией - использование системной шины, возникают ограничения, связанные с с-архитектурой: есть системные потоки служебной информации, что ведет к дополнительным накладным расходам - загрузка общей шины служебной информацией

• **Компьютеры с распределенной памятью.**

Состоит из вычислительных узлов, каждый из которых является полноценным компьютером со своей памятью, ОС, устройствами ввода-вывода и т.п., взаимодействующими друг с другом через коммуникационную среду. (-) сложность параллельного программирования; (+) относительная простота увеличения числа процессоров (рост производительности).

Основные понятия для распределенных систем

• **Длина критического пути** - минимальное количество элементарных связей, которые нужно пройти для коммутации двух самых удаленных процессоров.

• **Связность** - минимальное количество элементарных связей, которые нужно удалить, чтобы схема распалась на две несвязанные части.

• **Сложность** - общее количество необходимых элементарных связей.

• **Латентность и пропускная способность сети** - основные параметры коммуникационной сети кластеров.

Латентность - время начальной задержки при посылке сообщений.

Пропускная способность сети определяется скоростью передачи информации в единицу времени. Время на передачу сообщения в единицу времени вычисляется по следующей формуле: $t_N = t_0 + \frac{N}{S}$, где t_0 - латентность, N - объем передаваемых данных, S - пропускная способность сети.

• **Сообщения** - короткие фрагменты информации, передаваемые между процессорами.

• **Фрейм** - блок сообщений, состоящий из нескольких сообщений.

• **Блок** - блок, состоящий из нескольких фреймов.

• **Флек** - блок, состоящий из нескольких блоков.

• **Фрейм-переменные**, называемые также переменными экземплярами.

• **Члены-функции**, это просто функции.

• **Блоки-функции** - методы класса.

• **Параметры** - параметры класса.

• **Поля** - поля класса.

• **Функции** - методы класса.

• **Виртуальные функции** - методы класса.

• **Статические функции** - методы класса.

• **Виртуальные методы** - методы класса.

• **Сложные методы** - методы класса.

• **Методы наследования** - методы класса.

• **Сложные методы наследования</b**

OSN 25 Линейные обыкновенные дифференциальные уравнения и системы. Фундаментальная система решений. Определитель Бронского.

Линейным дифференциальным оператором n -го порядка называется оператор \mathcal{L} , линейным дифференциальным уравнением n -го порядка называется $f(t)$:

$\mathcal{L}y = f(t) = a_0(y^{(n)}(t) + a_1(y^{(n-1)}(t) + \dots + a_{n-1}(t)y'(t) + a_n(t)y(t))$, где $\mathcal{L}y(t) \in C[a, b]$, $f(t)$ – комплекснозначная функция, $a_k(t) \in C[a, b]$, $a_k(t) \in R$, $a_0(t) \neq 0$, $t \in [a, b]$,

$y(t) \in C^{(n)}[a, b]$ (n раз диф-ма на отрезке $[a, b]$),

Если $f(t) = 0$ на $[a, b]$, то уравнение называется однородным, иначе неоднородным.

Теорема 1: Если функции $y_k(t)$, $k=1..m$ являются решениями уравнений $\mathcal{L}y_k = f_k(t)$, то функция $y(t) = \sum_{k=1}^m c_k y_k(t)$, где c_k – комплексные постоянные, – является решением уравнения $\mathcal{L}y = f(t)$, где $f(t) = \sum_{k=1}^m c_k f_k(t)$.

▲ $\mathcal{L}y = \mathcal{L}(\sum_{k=1}^m c_k y_k(t)) = \sum_{k=1}^m c_k \mathcal{L}y_k(t) = \sum_{k=1}^m c_k f_k(t) = f(t)$, $t \in [a, b]$ ■

Следствие: Линейная комбинация решений однородного уравнения является решением однородного уравнения. Разность двух решений неоднородного уравнения с одинаковой правой частью есть решение однородного уравнения

Теорема 2 Решение задачи Коши $\mathcal{L}y = f(t)$, $y^{(k)}(t_0) = y_{0k}$, $k = 0, n-1$ представимо в виде $y(t) = v(t) + w(t)$, где функция $v(t)$ является решением задачи Коши для неоднородного уравнения $\mathcal{L}v = f(t)$ с нулевыми начальными условиями $v^{(k)}(t_0) = 0$, $k = 0, n-1$, а функция $w(t)$ является решением задачи Коши для однородного уравнения $\mathcal{L}w = 0$ с ненулевыми начальными условиями $w^{(k)}(t_0) = y_{0k}$, $k = 0, n-1$.

▲ Сумма $y(t) = v(t) + w(t)$ удовлетворяет неоднородному ур-ю в силу теоремы 1. Для начальных условий имеем равенства $y^{(k)}(t_0) = v^{(k)}(t_0) + w^{(k)}(t_0) = 0 + y_{0k} = y_{0k}$, $k = 1..n-1$ ■

Скалярные функции $\varphi_1(t), \dots, \varphi_m(t)$ называются линейно зависимыми на отрезке $[a, b]$, если найдутся такие комплексные константы

$$c_k \in \mathbb{C}, k = 1, m, \sum_{k=1}^m |c_k| > 0, \text{ что справедливо равенство } \sum_{k=1}^m c_k \varphi_k(t) = 0.$$

0, $\forall t \in [a, b]$. Если равенство выполнено только для $c_k = 0$, $k = 1..n$, то функции линейно независимы.

Определитель Бронского (вронкианом) системы функций $\varphi_1(t), \dots, \varphi_m(t)$, где $\varphi_i(t) \in C^{(m-1)}[a, b]$, называется зависящий от переменной $t \in [a, b]$ определитель

$$W[\varphi_1, \dots, \varphi_m](t) = \begin{vmatrix} \varphi_1(t) & \varphi_2(t) & \dots & \varphi_m(t) \\ \varphi_1'(t) & \varphi_2'(t) & \dots & \varphi_m'(t) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_1^{(m-1)}(t) & \varphi_2^{(m-1)}(t) & \dots & \varphi_m^{(m-1)}(t) \end{vmatrix}$$

Теорема 3: если система скалярных функций $\varphi_1(t), \dots, \varphi_m(t)$, где $\varphi_i(t) \in C^{(m-1)}[a, b]$, является линейно зависимой на отрезке $[a, b]$, то определитель Бронского этого системы тождественно равен нулю на этом отрезке: $W[\varphi_1, \dots, \varphi_m](t) \equiv 0, \forall t \in [a, b]$.

▲ Так как функции $\varphi_k(t)$ линейно зависимы на $[a, b]$, то существует нетривиальный набор констант c_1, \dots, c_m , для которого на отрезке $[a, b]$ справедливо равенство выше. В этом равенстве допустимо почленное дифференцирование до порядка $m-1$ включительно:

$$c_1\varphi_1^{(k)}(t) + \dots + c_m\varphi_m^{(k)}(t) = 0, k = 0, m-1, t \in [a, b].$$

Отсюда следует, что вектор-столбцы определителя Бронского линейно зависимы для всех $t \in [a, b]$. Следовательно, этот определитель равен нулю для всех $t \in [a, b]$. ■

Замечание. Из ЛНЗ не следует, что $W \neq 0$, контриимер: $\varphi_1(t) = t^2$, $\varphi_2(t) = -t^2$, $t < 0$; $t^2 \geq 0$, $W \equiv 0$, но на $[-1, 1]$ функции ЛНЗ.

Теорема 4: Для решений $y_1(t), \dots, y_n(t)$ линейного однородного уравнения на отрезке $[a, b]$ справедлива следующая альтернатива:

либо $W[y_1, \dots, y_n](t) = 0$ на $[a, b]$ и функции $y_1(t), \dots, y_n(t)$ линейно зависимы на этом отрезке;

либо $W[y_1, \dots, y_n](t) \neq 0, \forall t \in [a, b]$ и функции $y_1(t), \dots, y_n(t)$ линейно независимы на $[a, b]$.

Фундаментальной системой решений линейного однородного дифференциального уравнения n -го порядка на отрезке $[a, b]$ называется система из n линейно независимых на данном отрезке решений этого уравнения.

Общим решением линейного однородного (неоднородного) дифференциального уравнения n -го порядка называется зависящее от n произвольных постоянных решения этого уравнения, такое, что любое другое решение уравнения может быть получено из него в результате выбора некоторых значений этих постоянных.

Теорема 5: У любого линейного однородного уравнения $\mathcal{L}y = 0$ существует фундаментальная система решений на $[a, b]$.

▲ Рассмотрим постоянную матрицу B с элементами b_{ij} , $i, j = 1, 2, \dots, n$ такую, что $det B \neq 0$. Обозначим через $y_j(t)$ решения задачи Коши для уравнения $\mathcal{L}y = 0$ с начальными условиями:

$$y_j(t_0) = b_{1j}, y'_j(t_0) = b_{2j}, \dots, y_{n-1}(t_0) = b_{nj}, j = 1, 2, \dots, n.$$

По теореме существования и единственности решения задачи Коши для линейного дифференциального уравнения n -го порядка функции $y_j(t)$ существуют и определены однозначно. Составленный из них определитель Бронского $W[y_1, \dots, y_n](t_0)$, в силу начальных условий, таков, что $W[y_1, \dots, y_n](t_0) = det B \neq 0$. Следовательно, по предыдущей теореме он не равен нулю ни в одной точке отрезка $[a, b]$. Значит, они образуют фундаментальную систему решений уравнения $\mathcal{L}y = 0$. ■

Теорема 6: Пусть $y_1(t), y_2(t), \dots, y_n(t)$ – фундаментальная система решений линейного однородного уравнения уравнения $\mathcal{L}y = 0$ на отрезке $[a, b]$. Тогда общее решение этого уравнения на рассматриваемом отрезке имеет вид:

$$y_O(t) = c_1 y_1(t) + c_2 y_2(t) + \dots + c_n y_n(t), \quad \forall c_j \in \mathbb{C}. \quad (21)$$

▲ Так как линейная комбинация решений однородного уравнения $\mathcal{L}y = 0$ является решением этого уравнения, то при любых значениях постоянных c_k функция $y_O(t)$, определяемая формулой (21), является решением линейного однородного дифференциального уравнения $\mathcal{L}y = 0$. Покажем теперь, что любое решение уравнения $\mathcal{L}y = 0$ может быть получено из (21) в результате выбора значений постоянных c_k . Пусть $\tilde{y}(t)$ – некоторое решение уравнения $\mathcal{L}y = 0$. Рассмотрим систему алгебраических уравнений относительно неизвестных c_k :

$$\begin{cases} c_1 y_1(t_0) + c_2 y_2(t_0) + \dots + c_n y_n(t_0) = \tilde{y}(t_0) \\ c_1 y'_1(t_0) + c_2 y'_2(t_0) + \dots + c_n y'_n(t_0) = \tilde{y}'(t_0) \\ \dots \\ c_1 y_1^{(n-1)}(t_0) + c_2 y_2^{(n-1)}(t_0) + \dots + c_n y_n^{(n-1)}(t_0) = \tilde{y}^{(n-1)}(t_0) \end{cases} \quad (22)$$

Рассмотрим функцию $\hat{y}(t) = \sum_{k=1}^n \tilde{c}_k y_k(t)$. Эта функция является решением уравнения $\mathcal{L}y = 0$. Так как постоянные $\tilde{c}_1, \tilde{c}_2, \dots, \tilde{c}_n$ представляют собой решение системы (22), то функция $\hat{y}(t)$ такова, что $\hat{y}^{(k)}(t_0) = \tilde{y}^{(k)}(t_0)$, $k = 0, 1, \dots, n-1$.

Следовательно, функции $\hat{y}(t)$ и $\tilde{y}(t)$ являются решениями уравнения $\mathcal{L}y = 0$ и удовлетворяют одним и тем же начальными условиям в точке t_0 . По теореме о существовании и единственности решения задачи Коши эти функции совпадают: $\hat{y}(t) = \tilde{y}(t) = \sum_{k=1}^n \tilde{c}_k y_k(t)$ ■

Теорема 7: Пусть $y_1(t), y_2(t), \dots, y_n(t)$ – фундаментальная система решений линейного однородного уравнения $\mathcal{L}y = 0$ на отрезке $[a, b]$, $y_H(t)$ – некоторое (частное) решение неоднородного уравнения $\mathcal{L}y = f(t)$. Тогда общее решение линейного неоднородного уравнения $\mathcal{L}y = f(t)$ на рассматриваемом отрезке имеет вид:

$$y_O(t) = y_H(t) + y_O(t) = y_H(t) + c_1 y_1(t) + c_2 y_2(t) + \dots + c_n y_n(t), \quad (23)$$

где c_1, c_2, \dots, c_n – произвольные комплексные постоянные.

▲ Для любого набора констант $c_j \in \mathbb{C}$ формула (23) определяет решение линейного неоднородного уравнения $\mathcal{L}y = f(t)$ в силу линейности уравнения. Согласно определению общего решения осталось показать, что выбором констант в (23) можно получить любое наперед заданное решение $\mathcal{L}y = f(t)$, то есть для любого решения $\tilde{y}(t)$ неоднородного уравнения $\mathcal{L}y = f(t)$ найдутся константы $\tilde{c}_1, \tilde{c}_2, \dots, \tilde{c}_n$ такие, что на отрезке $[a, b]$ будет выполнено равенство

$$\tilde{y}(t) = y_H(t) + \tilde{c}_1 y_1(t) + \tilde{c}_2 y_2(t) + \dots + \tilde{c}_n y_n(t). \quad (24)$$

Пусть $\tilde{y}(t)$ – решение неоднородного уравнения $\mathcal{L}y = f(t)$. Разность $y(t) = \tilde{y}(t) - y_H(t)$ двух решений линейного неоднородного уравнения $\mathcal{L}y = f(t)$ является решением однородного уравнения $\mathcal{L}y = 0$. По теореме об общем решении линейного однородного уравнения найдутся комплексные константы \tilde{c}_j такие, что на рассматриваемом отрезке выполнено равенство $\tilde{y}(t) = \tilde{c}_1 y_1(t) + \tilde{c}_2 y_2(t) + \dots + \tilde{c}_n y_n(t)$, а вместе с ним и искомое равенство (24). ■

OSN 25 Линейные обыкновенные дифференциальные уравнения и системы. Фундаментальная система решений. Определитель Бронского.

Линейным дифференциальным оператором n -го порядка называется оператор \mathcal{L} , линейным дифференциальным уравнением n -го порядка называется $f(t)$:

$\mathcal{L}y = f(t) = a_0(y^{(n)}(t) + a_1(y^{(n-1)}(t) + \dots + a_{n-1}(t)y'(t) + a_n(t)y(t))$, где $y(t) \in C[a, b]$, $a_k(t) \in C[a, b]$, $a_k(t) \in R$, $a_0(t) \neq 0$, $t \in [a, b]$,

$y(t) \in C^{(n)}[a, b]$ (n раз диф-ма на отрезке $[a, b]$),

Если $f(t) = 0$ на $[a, b]$, то уравнение называется однородным, иначе неоднородным.

Теорема 1: Если функции $y_k(t)$, $k=1..m$ являются решениями уравнений $\mathcal{L}y_k = f_k(t)$, то функция $y(t) = \sum_{k=1}^m c_k y_k(t)$, где c_k – комплексные постоянные, – является решением уравнения $\mathcal{L}y = f(t)$, где $f(t) = \sum_{k=1}^m c_k f_k(t)$.

▲ $\mathcal{L}y = \mathcal{L}(\sum_{k=1}^m c_k y_k(t)) = \sum_{k=1}^m c_k \mathcal{L}y_k(t) = \sum_{k=1}^m c_k f_k(t) = f(t)$, $t \in [a, b]$ ■

Следствие: Линейная комбинация решений однородного уравнения является решением однородного уравнения. Разность двух решений неоднородного уравнения с одинаковой правой частью есть решение однородного уравнения

Теорема 2 Решение задачи Коши $\mathcal{L}y = f(t)$, $y^{(k)}(t_0) = y_{0k}$, $k = 0, n-1$ представимо в виде $y(t) = v(t) + w(t)$, где функция $v(t)$ является решением задачи Коши для неоднородного уравнения $\mathcal{L}v = f(t)$ с нулевыми начальными условиями $v^{(k)}(t_0) = 0$, $k = 0, n-1$, а функция $w(t)$ является решением задачи Коши для однородного уравнения $\mathcal{L}w = 0$ с ненулевыми начальными условиями $w^{(k)}(t_0) = y_{0k}$, $k = 0, n-1$.

▲ Сумма $y(t) = v(t) + w(t)$ удовлетворяет неоднородному ур-ю в силу теоремы 1. Для начальных условий имеем равенства $y^{(k)}(t_0) = v^{(k)}(t_0) + w^{(k)}(t_0) = 0 + y_{0k} = y_{0k}$, $k = 1..n-1$ ■

Скалярные функции $\varphi_1(t), \dots, \varphi_m(t)$ называются линейно зависимыми на отрезке $[a, b]$, если найдутся такие комплексные константы

$$c_k \in \mathbb{C}, k = 1, m, \sum_{k=1}^m |c_k| > 0, \text{ что справедливо равенство } \sum_{k=1}^m c_k \varphi_k(t) = 0.$$

0, $\forall t \in [a, b]$. Если равенство выполнено только для $c_k = 0$, $k = 1..n$, то функции линейно независимы.

Определитель Бронского (вронкианом) системы функций $\varphi_1(t), \dots, \varphi_m(t)$, где $\varphi_i(t) \in C^{(m-1)}[a, b]$, называется линейно зависимым на отрезке $[a, b]$, если найдутся такие комплексные константы

$$c_k \in \mathbb{C}, k = 1,$$

OSN 32 Численное решение задачи Коши для обыкновенных дифференциальных уравнений. Примеры методов Рунге–Кутта.

Рассматривается задача Коши для системы ОДУ:

$$\begin{cases} \frac{du}{dt} = f(t, u(t)), & t > 0, \\ u(0) = u_0, \end{cases} \quad (27)$$

где $u(t) = (u_1(t), \dots, u_m(t))^T$, $f(t, u(t)) = (f_1(t, u(t)), \dots, f_m(t, u(t)))^T$.

Обозначим $|u(t)| = \sqrt{u_1^2(t) + u_2^2(t) + \dots + u_m^2(t)}$.
Теорема: Пусть $f(t, u)$ непрерывна в параллелепипеде $R = \{|t| \leq a, |u(t) - u(0)| \leq b, a \in \mathbb{R}\}$ и удовлетворяет в R условию Липшица по второму аргументу, т.е. $|f(t, u) - f(t, v)| \leq L|u - v|$ для всех $(t, u), (t, v) \in R$
 \Rightarrow Эта задача (27), определенное и непрерывное на некотором отрезке.

В приведенных ниже примерах для простоты изложения предполагается, что система (27) состоит всего из одного уравнения.

Нахождение численного решения

Двухэтапный метод Рунге–Кутта

Общий вид двухэтапного метода Рунге–Кутта для уравнения (27):

$$\begin{cases} \frac{y_{n+1} - y_n}{\tau} = \sigma_1 K_1 + \sigma_2 K_2, & n \in \mathbb{Z}_+ \\ y_0 = u_0, \\ K_1 = f(t_n, y_n), \quad K_2 = f(t_n + a_2 \tau, y_n + b_2 \tau f(t_n, y_n)), \end{cases} \quad (28)$$

где $\sigma_1, \sigma_2, a_2, b_2 \in \mathbb{R}$ – некоторые числа, от выбора которых зависит как погрешность аппроксимации, так и точность численного решения. Подставим значения K_1 и K_2 в первое уравнение системы (28):

$$\frac{y_{n+1} - y_n}{\tau} = \sigma_1 f(t_n, y_n) + \sigma_2 f(t_n + a_2 \tau, y_n + b_2 \tau f(t_n, y_n)).$$

Рассмотрим погрешность аппроксимации разностной схемы (28) на решении задачи (27):

$$\psi_n = -\frac{u_{n+1} - u_n}{\tau} + \sigma_1 f(t_n, u_n) + \sigma_2 f(t_n + a_2 \tau, u_n + b_2 \tau f(t_n, u_n)). \quad (29)$$

Утв.: Погрешность аппроксимации этого метода имеет второй порядок малости по τ : $\psi_n = O(\tau^2)$ при $\sigma_2 = \sigma, \sigma_1 = 1 - \sigma, a_2 = b_2 = \sigma/2$.

Рядок u_{n+1} в ряд Тейлора в окрестности точки t_n :

$$\frac{u_{n+1} - u_n}{\tau} = u'_n + \frac{\tau}{2} u''_n + O(\tau^2).$$

Далее разложим $f(t_n + a_2 \tau, u_n + b_2 \tau f(t_n, u_n))$ в окрестности точки (t_n, u_n) :

$$(t_n + a_2 \tau, u_n + b_2 \tau f(t_n, u_n)) = f(t_n, u_n) + a_2 \tau \frac{\partial f}{\partial t} + b_2 \tau f(t_n, u_n) + O(\tau^2).$$

Причем $u' = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial u}$. Тогда погрешность аппроксимации примет вид:

$$u' = u'_n + (\sigma_1 + \sigma_2) f(t_n, u_n) + \tau ((a_2 \sigma_2 - 0.5) \frac{\partial f}{\partial t} + (b_2 \sigma_2 - 0.5) f_n) + O(\tau^2).$$

Чтобы получить оценку погрешности со вторым порядком по τ , необходимо избавиться от слагаемых, содержащих τ в первой степени. Для этого положим:

$$\sigma_1 + \sigma_2 = 1, \sigma_2 \sigma_2 = \sigma_2 b_2 = 0.5.$$

Тогда $\psi_n = O(\tau^2)$.

Погрешность решения разностной схемы (28): $z_n = y_n - u_n, n \in \mathbb{Z}$. Утв.: Общий двухэтапный метод Рунге–Кутта при выполнении соответствующих условий имеет квадратичную точность по τ , т.е. при достаточно малых τ : $|z_{n+1}| = O(\tau^2)$, совпадающую с оценкой погрешности аппроксимации на решении исходного уравнения (27).

Частные случаи общего двухэтапного метода Рунге–Кутта:

1. При $\sigma = 1, a = a_2 = 0.5, b = b_2 = 0.5$ мы получим схему Рунге–Кутта «предиктор–корректор»: $y_{n+1} = y_n + \tau f(t_{n+\frac{1}{2}}, y_n + 0.5 \tau f(t_n, y_n))$. Погрешность этой схемы равна $O(\tau^2)$.

2. Если положить $\sigma = 0.5, a = 1, b = 1$, то мы получим симметричную разностную схему:

$$\frac{y_{n+1} - y_n}{\tau} = 0.5 (f(t_n, y_n) + f(t_n + \tau, y_n + \tau f_n)), n \in \mathbb{Z}_+, y_0 = u_0.$$

Эта разностная схема является очень эффективной, имеет второй порядок точности по τ и часто используется на практике.

Общий m -этапный метод Рунге–Кутта

Общая идея m -этапного метода Рунге–Кутта заключается в том, что для вычисления значения приближенного решения в каждой следующей точке t_{n+1} вводятся m дополнительных этапов. Промежуточные значения на каждом шаге $n \in \mathbb{Z}_+$ вычисляются по формулам:

$$\begin{aligned} K_1 &= f(t_n, y_n), \\ K_2 &= f(t_n + a_2 \tau, y_n + b_2 \tau K_1), \\ K_3 &= f(t_n + a_3 \tau, y_n + b_3 \tau K_1 + b_{31} \tau K_2), \\ &\dots \\ K_m &= f(t_n + a_m \tau, y_n + b_{m1} \tau K_1 + b_{m2} \tau K_2 + \dots + b_{m(m-1)} \tau K_{m-1}). \end{aligned}$$

При этом разностная схема для исходной задачи (27) имеет вид

$$\begin{cases} \frac{y_{n+1} - y_n}{\tau} = \sigma_1 K_1 + \sigma_2 K_2 + \dots + \sigma_m K_m \\ y_0 = u_0, n \in \mathbb{Z}_+, \end{cases} \quad (30)$$

где $\sigma_1, \dots, \sigma_m \in \mathbb{R}$, и выполнено условие аппроксимации: $\sum_{i=1}^m \sigma_i = 1$.

Примеры трех- и четырех-этапных методов Рунге–Кутта, имеющих третий и четвертый порядок точности соответственно:

$$1. m = 3: \frac{y_{n+1} - y_n}{\tau} = \frac{1}{6} (K_1 + 4K_2 + K_3), \text{ где}$$

$$K_1 = f(t_n, y_n),$$

$$K_2 = f(t_n + 0.5\tau, y_n + 0.5\tau K_1),$$

$$K_3 = f(t_n + \tau, y_n - \tau K_1 + 2\tau K_2).$$

Данная схема имеет третий порядок точности по τ : $O(\tau^3)$.

Замечание: Формулы m -этапного метода Рунге–Кутта достаточно громоздки. Это является одной из причин того, что на практике редко используются методы Рунге–Кутта для $m > 4$.

[Ионкин, *Лекции по курсу Численные методы*, page 152-163]

OSN 30 Квадратурные формулы прямоугольников, трапеций и парабол.

Задача: Вычислить определенный интеграл $I = \int_a^b f(x) dx$.

Не всегда можно посчитать аналитически, но есть универсальные алгоритмы – формулы численного интегрирования (квадратурные формулы). В них интеграл заменяется конечной суммой:

$\int_a^b f(x) dx \approx \sum_{k=0}^N C_k f(x_k)$ – квадратурная формула, C_k – коэффициенты квадратурной формулы, $x_k \in [a, b]$ – узлы квадратурной формулы.

$\psi = \int_a^b f(x) dx \sim \sum_{k=0}^N C_k f(x_k)$ – погрешность квадратурной формулы.

Введем на $[a, b]$ равномерную сетку $\omega_n = \{x_i = a + ih, i \in [0, N], N_h = b - a\}$.

Итак, $\int_a^b f(x) dx \sim \sum_{i=0}^{x_i} \int_{x_{i-1}}^{x_i} f(x) dx$. Строим квадратурные формулы для $\int_{x_{i-1}}^{x_i} f(x) dx$.

Формула прямоугольников.

$\int_{x_{i-1}}^{x_i} f(x) dx \sim f\left(x_{i-\frac{1}{2}}\right) h$

$\psi_i = \int_{x_{i-1}}^{x_i} f(x) dx - f\left(x_{i-\frac{1}{2}}\right) h = \int_{x_{i-1}}^{x_i} (f(x) - f\left(x_{i-\frac{1}{2}}\right)) dx =$

$= \int_{x_{i-1}}^{x_i} \left(f\left(x_{i-\frac{1}{2}}\right) + (x - x_{i-\frac{1}{2}}) f'(x_{i-\frac{1}{2}}) + \frac{(x - x_{i-\frac{1}{2}})^2}{2} f''(\xi) \right) dx$

$= |\psi_i| \leq M_{2i} \int_{x_{i-1}}^{x_i} \frac{(x - x_{i-\frac{1}{2}})^2}{2} dx = \frac{h^3}{24} M_{2i}$, где $M_{2i} = \max_{x \in [x_{i-1}; x_i]} |f''(x)|$

$\int_a^b f(x) dx \sim \sum_{i=0}^N f(x_{i-\frac{1}{2}}) h$

$\psi = \sum_{i=0}^N \psi_i \leq \sum_{i=0}^N \frac{h^3}{24} M_{2i} \leq \frac{M_2 N h^3}{24} = \frac{M_2 h^2 (b - a)}{24} \Rightarrow \psi = O(h^2)$

Формула трапеций.

$\int_{x_{i-1}}^{x_i} f(x) dx \sim \frac{f(x_{i-1}) + f(x_i)}{2} h$

получается путем замены $f(x)$ интерполяционным многочленом первой степени, построенным из узлов x_{i-1}, x_i .

$$L_{1i} = \frac{(x - x_{i-1}) f(x_i) - (x - x_i) f(x_{i-1})}{h}$$

$$f(x) - L_{1i} = \frac{(x - x_{i-1})(x - x_i)}{2} f''(\xi_i)$$

$|\psi_j| = \int_{x_{i-1}}^{x_i} f(x) dx - \frac{f(x_{i-1}) + f(x_i)}{2} h = \int_{x_{i-1}}^{x_i} (f(x) - L_{1i}) dx =$

$= \int_{x_{i-1}}^{x_i} \frac{(x - x_{i-1})(x - x_i)}{2} f''(\xi_i) \Rightarrow |\psi_j| \leq \frac{M_3 j h^3}{12}$

$\int_a^b f(x) dx \sim \sum_{i=1}^N \frac{f(x_{i-1}) + f(x_i)}{2} h =$

$= h (0.5 f(x_0) + f(x_1) + f(x_2) + \dots + f(x_{N-1}) + 0.5 f(x_N))$ – составная формула трапеций.

$$|\psi| \leq \frac{M_3 h^2 (b - a)}{12} = O(h^2)$$

Формула Симпсона (парабол).

$L_n = \sum_{k=0}^n L_{nk} = \sum_{k=0}^n \frac{\omega(x)}{(x - x_k) \omega'(x_k)} f(x_k)$ – интерполяционный полином в Форме Лагранжа, где

$$\omega(x) = \prod_{j=0}^n (x - x_j), \quad \omega'(x_k) = \prod_{j=0, j \neq k}^n (x_k - x_j), \quad f(x) - L_n = \frac{f^{(n+1)}(\xi(x))}{(n+1)!} \omega(x)$$

В формуле Симпсона:

$$f(x) \sim L_2 \sim \frac{(x - x_{i-\frac{1}{2}})(x - x_i)}{2} f(x_{i-\frac{1}{2}}) +$$

$$= \frac{(x - x_{i-1})(x - x_i)}{2} f(x_{i-\frac{1}{2}}) + \frac{(x - x_{i-1})(x - x_{i-\frac{1}{2}})}{2} f(x_i) =$$

$$= \frac{2}{h^2} ((x - x_{i-\frac{1}{2}})(x - x_i) f(x_{i-\frac{1}{2}}) - 2(x - x_{i-1})(x - x_i) f(x_{i-\frac{1}{2}}) + (x - x_{i-1})(x - x_{i-\frac{1}{2}}) f(x_i)), \forall x \in [x_{i-1}, x_i]$$

$$\int_{x_{i-1}}^{x_i} L_2(x) dx = \frac{h}{6} (f(x_{i-1}) + 4f(x_{i-\frac{1}{2}}) + f(x_i))$$

$$\Rightarrow \int_a^b f(x) dx \approx \sum_{i=1}^N \frac{h}{6} (f(x_{i-1}) + 4f(x_{i-\frac{1}{2}}) + f(x_i))$$

$$\psi_i \leq \frac{M_4 h^4}{2880}, |\psi| \leq \frac{M_4 h^4 (b - a)}{2880} \Rightarrow \psi = O(h^4)$$

Сюда бы картинки разбили под кривой для разных методов

[Пупкин-Залупкин, *Напиши источник!*, page 69-96]

OSN 28 Схемы из функциональных элементов и простейшие алгоритмы их синтеза. Оценка сложности схем, получаемых по методу Шенниона.

Орграф – это ориентированный граф.

Вершины орграфа, в которые не входит ни одной дуги, называются **истоками**.

Орграф называется **ациклическим**, если в нем нет ориентированных циклов.

Система Б = g_1, g_2, \dots, g_m , где все g_i – функции алгебры логики, будем называть **базисом функциональных элементов**.

Орграф называется **упорядоченным**, если для каждой вершини v_i , в которую входит k_i дуг, задан порядок e_1, e_2, \dots, e_{k_i} этих дуг.

Схемой из функциональных элементов в базисе Б называется ациклический упорядоченный орграф, в котором:

- каждому истоку присвоена некоторая переменная, причем разным истокам присвоены разные переменные (истоки при этом называются входами схемы, а присво

OSN 33 Задача Коши для уравнения колебания струны. Формула Даламбера.

Задача Коши для уравнения колебания струны.

$$\begin{cases} u_{tt} = a^2 u_{xx} + f(x, t) \\ u(x, 0) = \varphi(x) \\ u_t(x, 0) = \psi(x) \end{cases}$$

где $t > 0$, $a > 0$, $u(x, t) \in C^2(t > 0, x \in \mathbb{R}) \cap C^1(t \geq 0, x \in \mathbb{R})$.

Физическая интерпретация: уравнение малых поперечных колебаний струны. $u(x, t)$ – положение точки струны с координатой x в момент времени t . Если концы струны закреплены, то $u(0, t) = 0, u(l, t) = 0$. Так как процесс колебаний струны зависит от ее начальной формы и распределения скоростей, то задают начальные условия: $a = \sqrt{\frac{T_0}{\rho}}$, где

T_0 – величина натяжения (не зависит от x , ρ – линейная плотность струны. $f(x, t)$ – плотность внешних сил.

Далее будем рассматривать $f(x, t) = 0$.

(Представим что) $\frac{\partial^2 u}{\partial t^2} = \frac{d^2 u}{dt^2}$, $u \frac{\partial^2 u}{\partial x^2} = \frac{d^2 u}{dx^2}$: $\frac{d^2 u}{dt^2} = a^2 \frac{d^2 u}{dx^2} \Rightarrow d^2 u dx^2 = a^2 dt^2 d^2 u \Rightarrow dx^2 = a^2 dt^2$) Характеристическое уравнение: $dx^2 - a^2 dt^2 = 0$.

$dx - adt = 0$, $dx + adt = 0 \Rightarrow x - at = C_1 = const$, $x + at = C_2 = const$

Сделаем замену переменных:

$$x + at = \xi, \quad x - at = \eta$$

$$\xi_x = 1, \quad \xi_{xx} = 0, \quad \eta_x = 1, \quad \eta_{xx} = 0,$$

$$\xi_t = a, \quad \xi_{tt} = 0, \quad \eta_t = -a, \quad \eta_{tt} = 0.$$

Тогда:

$$u_x = u_\xi \cdot \xi_x + u_\eta \cdot \eta_x,$$

$$u_t = u_\xi \cdot \xi_t + u_\eta \cdot \eta_t,$$

$$u_{xx} = u_{\xi\xi} \cdot \xi_x^2 + 2u_{\xi\eta} \cdot \xi_x \cdot \eta_x + u_\xi \cdot \xi_{xx} + u_{\eta\eta} \cdot \eta_x^2 + u_\eta \cdot \eta_{xx},$$

$$u_{tt} = u_{\xi\xi} \cdot \xi_t^2 + 2u_{\xi\eta} \cdot \xi_t \cdot \eta_t + u_\xi \cdot \xi_{tt} + u_{\eta\eta} \cdot \eta_t^2 + u_\eta \cdot \eta_{tt}.$$

Подставляем в уравнение $u_{tt} = a^2 u_{xx}$:

$$\begin{aligned} u_{\xi\xi} \cdot \xi_t^2 + 2u_{\xi\eta} \cdot \xi_t \cdot \eta_t + \underbrace{u_\xi \cdot \xi_{tt}}_{=0} + u_{\eta\eta} \cdot \eta_t^2 + \underbrace{u_\eta \cdot \eta_{tt}}_{=0} = \\ = a^2(u_{\xi\xi} \cdot \xi_x^2 + 2u_{\xi\eta} \cdot \xi_x \cdot \eta_x + \underbrace{u_\xi \cdot \xi_{xx}}_{=0} + u_{\eta\eta} \cdot \eta_x^2 + \underbrace{u_\eta \cdot \eta_{xx}}_{=0}) \end{aligned}$$

Преобразуем:

$$a^2 u_{\xi\xi} - 2a^2 u_{\xi\eta} + a^2 u_{\eta\eta} = a^2 u_{\xi\xi} + 2a^2 u_{\xi\eta} + a^2 u_{\eta\eta}$$

$$4a^2 u_{\xi\eta} = 0, a > 0$$

Получаем: $u_{\xi\eta} = 0$

Найдем общий интеграл этого уравнения: $u_\eta(\xi, \eta) = f^*(\eta)$, где $f^*(\eta)$ – некоторая функция только переменного η .

Интегрируя это равенство по η при фиксированном ξ , получим:

$$u(\xi, \eta) = \int f^*(\eta) d\eta = f_1(\xi) + f_2(\eta) \quad (35)$$

Обратно, каковы бы ни были дважды дифференцируемые функции f_1 и f_2 , функция $u(\xi, \eta)$, определяемая формулой (35), представляет собой решение уравнения $u_{\xi\eta} = 0$. Так как високое решение уравнения $u_{\xi\eta} = 0$ может быть представлено в виде (35) при соответствующем выборе f_1 и f_2 , то формула (35) является общим интегралом этого уравнения.

Следовательно, функция

$$u(x, t) = f_1(x + at) + f_2(x - at)$$

является общим интегралом уравнения $u_{tt} = a^2 u_{xx}$.

Удовлетворим начальными условиям:

$$\begin{cases} u(x, 0) = f_1(x) + f_2(0) = \varphi(x) \\ u_t(x, 0) = -af'_1(x) + af'_2(0) = \psi(x) \end{cases}$$

Проинтегрируем второе равенство, получим:

$$\begin{cases} f_1(x) + f_2(x) = \varphi(x) \\ f_1(x) - f_2(x) = \frac{1}{a} \int_{x_0}^x \psi(\alpha) d\alpha + C \end{cases}$$

Сложим и вычтем два полученных равенства:

$$\begin{cases} f_1(x) = \frac{1}{2} \varphi(x) + \frac{1}{2a} \int_{x_0}^x \psi(\alpha) d\alpha + \frac{C}{2} \\ f_2(x) = \frac{1}{2} \varphi(x) - \frac{1}{2a} \int_{x_0}^x \psi(\alpha) d\alpha - \frac{C}{2} \end{cases}$$

Подставим в $u(\xi, \eta) = f_1(x + at) + f_2(x - at)$ найденные выражения для f_1, f_2 :

$$u(x, t) = \frac{\varphi(x + at) + \varphi(x - at)}{2} + \frac{1}{2a} \left(\int_{x_0}^{x+at} \psi(\alpha) d\alpha - \int_{x_0}^{x-at} \psi(\alpha) d\alpha \right)$$

Окончательно:

$$u(x, t) = \frac{\varphi(x + at) + \varphi(x - at)}{2} + \frac{1}{2a} \int_{x-at}^{x+at} \psi(\alpha) d\alpha$$

Полученная формула – **формула Даламбера**.

Теорема о применимости формулы Даламбера. Пусть $\varphi(x) \in C^2(-\infty, \infty)$, $\psi \in C^1[0, +\infty)$, $a > 0$. Тогда формула Даламбера представляет единственное решение задачи Коши для уравнения колебания струны.

[А. Н. Тихонов, *Уравнения математической физики*, page 50-52]

OSN 34 Постановка краевых задач для уравнения теплопроводности. Метод разделения переменных для решения первой краевой задачи.

Краевые задачи для уравнения теплопроводности представляют собой математические модели процессов распространения тепла, например, в стержнях.

$u(x, t)$ – температура в сегменте с координатами x во время t .

$F(x, t)$ – плотность тепловых источников, $a^2 = \frac{k}{c\rho}$ – коэффициент температуропроводности, $f(x, t) = \frac{F(x, t)}{c\rho}$, c – удельная теплоемкость, k – коэффициент теплопроводности, ρ – плотность.

Одномерное уравнение теплопроводности:

$$u_t(x, t) = a^2 u_{xx}(x, t) + f(x, t)$$

Основные типы задач:

• **Первая краевая задача.**

$$\begin{cases} u_t(x, t) = a^2 u_{xx}(x, t) + f(x, t), \quad 0 < x < l, \quad t > 0 \\ u(0, t) = \mu_1(t), \quad t \geq 0 \\ u(l, t) = \mu_2(t) \\ u(x, 0) = \varphi(x), \quad 0 \leq x \leq l \end{cases}$$

• **Вторая краевая задача.**

$$\begin{cases} u_t(x, t) = a^2 u_{xx}(x, t) + f(x, t), \quad 0 < x < l, \quad t > 0 \\ u_x(0, t) = \nu_1(t), \quad t \geq 0 \\ u_x(l, t) = \nu_2(t) \\ u(x, 0) = \varphi(x), \quad 0 \leq x \leq l \end{cases}$$

• **Смешанная краевая задача.** – одно из краевых условий задано функцией $u(0, t)$ или $u(l, t)$, а другое производной u_x по x .

$u(x, t)$ – **решение 1-ой краевой задачи** для уравнения теплопроводности, если:

1. $u(x, t) \in C([0, l] \times [0, +\infty))$;
2. $u(x, t) \in C^2((0, l) \times (0, +\infty))$;
3. $u(x, t)$ удовлетворяет условиям 1-ой краевой задачи.

Далее будем рассматривать $f(x, t) = 0$.

Метод разделения переменных для решения первой краевой задачи.

$$\begin{cases} u_t(x, t) = a^2 u_{xx}(x, t), \quad 0 < x < l, \quad t > 0 \\ u(0, t) = 0, \quad t \geq 0 \\ u(l, t) = 0 \\ u(x, 0) = \varphi(x), \quad 0 \leq x \leq l \end{cases}$$

Будем искать решение в виде $u(x, t) = X(x)T(t)$. Рассмотрим задачу с однородными начальными и краевыми условиями:

$$\begin{cases} XT' = a^2 X'' T \\ X(0)T(0) = 0 \\ X(l)T(l) = 0 \end{cases} \rightarrow \begin{cases} \frac{T'}{T} = \frac{X''}{X} = -\lambda \\ X(0) = 0 \\ X(l) = 0 \end{cases}$$

Получаем две задачи:

1. Задача Штурма-Лиувилля:

$$\begin{cases} X'' + \lambda X = 0 \\ X(0) = X(l) = 0 \end{cases}$$

Рассматриваем 3 случая: $\lambda < 0$, $\lambda = 0$, $\lambda > 0$. При $\lambda > 0$ получаем $\lambda_n = (\frac{\pi n}{l})^2$, $X_n(x) = \sin(\frac{\pi n x}{l})$, $n \in \mathbb{N}$

2. $T' + (\frac{\pi n a}{l})^2 T = 0$

$$\text{Решение: } T_n(t) = a_n e^{-(\frac{\pi n a}{l})^2 t}, \quad n \in \mathbb{N}$$

Получаем решение: $u(x, t) = \sum_{n=1}^{\infty} a_n e^{-(\frac{\pi n a}{l})^2 t} \sin(\frac{\pi n x}{l})$

Для нахождения a_n используем начальное условие. Так как $\{\sin(\frac{\pi n x}{l})\}$ – замкнутая полная система функций, то \forall кусочно-дифференцируемую функцию можно разложить в ряд Фурье:

$$\varphi(x) = \sum_{n=1}^{\infty} \varphi_n \sin(\frac{\pi n x}{l}), \quad \varphi_n = \frac{2}{l} \int_0^l \varphi(\xi) \sin(\frac{\pi n \xi}{l}) d\xi$$

Так как $u(x, 0) = \varphi(x)$, получаем $a_n = \varphi_n$, $n \in \mathbb{N}$. Таким образом, получаем решение:

$$u(x, t) = \sum_{n=1}^{\infty} \frac{2}{l} \int_0^l \varphi(\xi) \sin\left(\frac{\pi n \xi}{l}\right) d\xi \cdot e^{-(\frac{\pi n a}{l})^2 t} \sin\left(\frac{\pi n x}{l}\right)$$

Для существования решения достаточно потребовать, чтобы $\varphi \in C^2[0, l]$ и $\varphi(0) = \varphi(l) = 0$.

Остальные случаи:

• $X'(0) = X(l) = 0$: $\lambda_n = \left(\frac{\pi(2n+1)}{2l}\right)^2$, $X_n = \cos(\sqrt{\lambda_n} x)$, $n = 0, 1, 2, \dots$

• $X(0) = X'(l) = 0$: $\lambda_n = \left(\frac{\pi(2n+1)}{2l}\right)^2$, $X_n = \sin(\sqrt{\lambda_n} x)$, $n = 0, 1, 2, \dots$

• $X'(0) = X'(l) = 0$: $\lambda_n = \left(\frac{\pi n}{l}\right)^2$, $X_n = \cos(\sqrt{\lambda_n} x)$, $n = 1, 2, \dots$; $X_0 = 1$

[А. Н. Тихонов, *Уравнения математической физики*, page 200-202]

DOP 1 Теорема Поста о полноте систем функций в алгебре логики.

Полная система ($\in P_2$) — множество A ФАЛ такое, что любую ФАЛ можно выразить формулой над A .

$\sqsubseteq A \subseteq P_2$. Тогда замыкание A (обозн. $[A]$) — множество всех ФАЛ, которые можно выразить формулами над A .

Класс A называется замкнутым, если $A = [A]$.

Говорят, что набор $\bar{\alpha} = (\alpha_1, \dots, \alpha_n)$ предшествует набору $\bar{\beta} = (\beta_1, \dots, \beta_n)$, $\bar{\alpha} \preceq \bar{\beta}$ ($\bar{\alpha}$ не больше $\bar{\beta}$), если $\alpha_i \leq \beta_i$ для всех $i = 1, \dots, n$.

Если $\bar{\alpha} \preceq \bar{\beta}$ и $\bar{\alpha} \neq \bar{\beta}$, то говорят, что набор $\bar{\alpha}$ строго предшествует набору $\bar{\beta}$, $\bar{\alpha} < \bar{\beta}$.

Наборы $\bar{\alpha}$ и $\bar{\beta}$ называются сравнимыми, если $\bar{\alpha} \preceq \bar{\beta}$ либо $\bar{\beta} \preceq \bar{\alpha}$.

Например, векторы $[0, 1]$ и $[1, 0]$ — несравнимы.

Стандартные замкнутые классы:

$- T_0 = \{f \in P_2 | f(0, \dots, 0) = 0\}$; — сохраняющие 0

$- T_1 = \{f \in P_2 | f(1, \dots, 1) = 1\}$; — сохраняющие 1

$- L = \{f(x_1, \dots, x_n) = a_0 + a_1x_1 + \dots + a_nx_n\}$ — линейные функции;

$- S = \{f(x_1, \dots, x_n) = f^*(x_1, \dots, x_n) = \bar{f}(x_1, \dots, x_n)\}$ — самодвойственные функции;

$- M = \{\alpha \leq \beta \rightarrow f(\alpha) \leq f(\beta)\}$ — монотонные функции;

Вспомогательные леммы:

1. Если булева функция f немонотона, то из неё подстановкой вместо аргументов констант 0 и 1 и переменной x можно получить \bar{x} .

▲ Немонотона $\Rightarrow \exists$ два набора $\alpha < \beta$, $a(f(\alpha)) = 1 > f(\beta) = 0$. Будем заменять в α 0 на 1 по одному, чтобы получился β , промежуточные назовём α_k , $\alpha_0 = \alpha$, $\alpha_r = \beta$. В какой-то момент получим $f(\alpha_k) = 1$, $f(\alpha_{k+1}) = 0$, получим два соседних набора α_k и α_{k+1} . Пусть различаются в i -й переменной. Заменим в α_k i -ую переменную на x , получим $\bar{\alpha}_k$, $f(\bar{\alpha}_k) = \bar{x}$.

2. Если булева функция несамодвойственна, то из неё подстановкой вместо аргументов переменной x и её отрицания \bar{x} можно получить либо константу 0, либо константу 1.

▲ Несамодвойственна, то \exists т.ч. $f(\alpha) = f(\bar{\alpha}) = C$. Рассмотрим $\phi(x) = f(x \oplus \alpha_1, \dots, x \oplus \alpha_n)$. Тогда в зависимости от x в аргументах функции набор α или $\bar{\alpha}$, в любом случае $\phi(x) = C$.

3. Если булева функция линейная, то из неё подстановкой вместо аргументов констант, переменных x , y , их отрицаний \bar{x} , \bar{y} можно получить $x \cdot y$ или $\bar{x} \cdot \bar{y}$.

▲ Рассмотрим полином J егалкина P_f функции f (представление в виде \oplus из конъюнкций). В нем найдется слагаемое, которое конъюнкциями двух или более переменных, пусть $x_1 \cdot x_2 \cdots \cdot x_r$.

$P_f = x_1 \cdot x_2 \cdot g_1(x_3, \dots, x_n) \oplus x_1 \cdot g_2(x_3, \dots, x_n) \oplus x_2 \cdot g_3(x_3, \dots, x_n) \oplus g_4(x_3, \dots, x_n)$

Либо $g_1 = 1$ (если $r = 2$), либо $\exists \alpha$ т.ч. $g_1(\alpha) = 1$. Обозначим $g_2(\alpha) = a$, $g_3(\alpha) = b$, $g_4(\alpha) = c$.

$\phi(x, y) = f(x \oplus b, y \oplus a, a_3, \dots, a_n) = (x \oplus b)(y \oplus a) \oplus a(x \oplus b) \oplus b(y \oplus a) \oplus c = \{раскроите сами\} xy \oplus d$, d -константа.

Теорема Поста. Система ФАЛ $A = \{f_1, f_2, \dots\}$ является полной в P_2 \iff она не содержитя целиком ни в одном из следующих классов: T_0 , T_1 , S , L , M .

▲ Необходимость. Пусть A — полная система, N — любой из классов T_0 , T_1 , S , L , M , и (от противного) пусть $A \subseteq N$. Тогда $[A] \subseteq [N] = N \neq P_2$, то есть $[A] \neq P_2$. Полученное противоречие завершает обоснование необходимости.

Достаточность. Пусть A не является подмножеством ни одного из этих классов. Тогда в A существуют функции $f_0 \notin T_0$, $f_1 \notin T_1$, $f_L \notin L$, $f_M \notin M$, $f_S \notin S$. Пусть $B = \{f_0, f_1, f_M, f_L, f_S\}$. Достаточно показать, что $[A] \supseteq [B] = P_2$.

Выразим формулами над B все функции из полной системы $\{0, 1, \bar{x}, x\}$.

• Получение отрицания. Рассмотрим функцию $f_0(x_1, \dots, x_n) \notin T_0$ и введём функцию $\varphi_0(x) = f_0(x, x, \dots, x)$. Так как функция f_0 не хранит нуль, $\varphi_0(0, 0, \dots, 0) = 1$. Возможны два случая: либо $\varphi_0(x) = \bar{x}$, либо $\varphi_0(x) \equiv 1$.

Рассмотрим функцию $f_1(x_1, \dots, x_n) \notin T_1$ и введём функцию $\varphi_1(x) = f_1(x, x, \dots, x)$. Так как функция f_1 не сохраняет единицу, $\varphi_1(1) = f_1(1, 1, \dots, 1) = 0$. Возможны два случая: либо $\varphi_1(x) = \bar{x}$, либо $\varphi_1(x) \equiv 1$.

Если хотя бы в одном случае получилось искомое отрицание, пункт завершен. Если же в обоих случаях получились константы, то согласно лемме о немонотонной функции, подставляя в функцию вместо всех переменных константы и тождественные функции, можно получить отрицание.

Отрицание получено.

• Получение констант 0 и 1. Имеем $f_S \notin S$. Согласно лемме о несамодвойственной функции, подставляя вместо всех переменных формулы f_S отрицание (которое получено в пункте 1) и тождественную функцию, можно получить константы: $[f_S, \bar{x}] \supseteq [0, 1]$.

Константы получены.

• Получение конъюнкции. Имеем функцию $f_L \notin L$. Согласно лемме о линейной функции, подставляя в функцию вместо всех переменных константы и отрицания (которые были получены на предыдущих шагах доказательства), можно получить либо конъюнкцию, либо отрицание конъюнкции. Однако на первом этапе отрицание уже получено, следовательно, всегда можно получить конъюнкцию: $[f_L, 0, 1, \bar{x}] \supseteq [x \cdot y, \bar{x} \cdot \bar{y}]$.

Конъюнкция получена.

В результате получено, что $[f_0, f_1, f_M, f_L, f_S] \supseteq [0, 1, \bar{x}, x \cdot y] = P_2$. Что и требовалось доказать. ■

[Презентации к госам от маткиба]

DOP 3 Логика 1-го порядка. Вполнимость и общезначимость. Общая схема метода резолюций.

Базовые символы:

- Предметные переменные $Var = \{x_1, x_2, \dots, x_k, \dots\}$;
- Предметные константы $Const = \{c_1, c_2, \dots, c_t, \dots\}$;
- Функциональные символы $Func = \{f_1^{n_1}, f_2^{n_2}, \dots, f_r^{n_r}, \dots\}$;
- Предикатные символы $Pred = \{P_1^{m_1}, P_2^{m_2}, \dots, P_s^{m_s}, \dots\}$.

Тройка $(Const, Pred, Func)$ называется сигнатурой алфавита.

Логические связи и кванторы:

Конъюнкция — \wedge

Дизъюнкция — \vee

Отрицание — \neg

Импликация — \rightarrow

Квантор всеобщности — \forall

Квантор существования — \exists

Определение терма: Терм — это

x , если $x \in Var$, x — переменная;
 c , если $c \in Const$, c — константа;

$f^n(t_1, t_2, \dots, t_n)$, если $f^n \in Func$, t_1, t_2, \dots, t_n — термы, — составной терм.

Терм — множество термов заданного алфавита. Var_t — множество переменных, входящих в состав терма.

$\{x_1, x_2, \dots, x_n\}$ — запись обозначающая терм t , у которого $Var_t \subseteq \{x_1, x_2, \dots, x_n\}$.

Формула — это либо атомарная формула $P_m(t_1, t_2, \dots, t_m)$, если $P_m \in Pred$, $\{t_1, t_2, \dots, t_m\} \subseteq Term$; либо составная формула $\phi, \psi \vee \phi$ и т.д., если ψ, ϕ — формулы;

Квантор связывает ту переменную, которая следует за ним. Вхождение переменной в области действия квантора, **связывающего** эту переменную, называется связанным. Вхождение переменной в формулу, не являющейся связанным, называется свободным. Переменная называется свободной, если она имеет свободное вхождение в формулу. $w(x_1, x_2, \dots, x_n)$ — запись, обозначающая формулу ψ , у которой $Var_\psi \subseteq \{x_1, x_2, \dots, x_n\}$. Если $Var_\psi = 0$, то формула ψ называется замкнутой формулой, или предложением . $CForm$ — множество всех замкнутых формул, или предложений из полной системы из $\{x_1, x_2, \dots, x_n\}$.

Интерпретация сигнатуры — это $(D_I, Const, Func, Pred)$, где

• D_I — неупорядоченное множество, которое называется областью интерпретации, предметной областью, универсумом.

• $Const$ — оценка констант — сопоставляет каждой константе предмет из области интерпретации.

• $Func$ — оценка функциональных символов — сопоставляет n -местной функции f функцию из области интерпретации.

• $Pred$ — оценка предикатных символов — сопоставляет каждому предикатному символу отношение из области интерпретации.

Описание выполнимости обозначается как \models . Пусть I — интерпретация.

$I \models \varphi(x_1, \dots, x_n)[d_1, \dots, d_n]$, т.е. формула выполнима в интерпретации I на наборе d_1, \dots, d_n , если

1. $\varphi(x_1, \dots, x_n) = P(t_1, \dots, t_m)$ и $\bar{P}(t_1[d_1, \dots, d_n], \dots, t_m[d_1, \dots, d_n]) = True$,

2. φ имеет вид $\psi_1 \wedge \psi_2$ и обе формулы выполнимы на наборе,

3. φ имеет вид $\psi_1 \vee \psi_2$ и хотя бы одна из формул выполнима на наборе,

4. φ имеет вид $\psi_1 \rightarrow \psi_2$ и на наборе либо выполнима ψ_2 , либо невыполнима ψ_1 ,

5. $\neg \varphi$ невыполнима на наборе,

6. если $\varphi(x_1, \dots, x_n) = \exists x_0 \psi(x_0, x_1, \dots, x_n)$ и для некоторого элемента d_0 , $d_0 \in D_I$ имеет место $I \models \psi(x_0, \dots, x_n)[d_1, \dots, d_n]$,

7. если $\varphi(x_1, \dots, x_n) = \forall x_0 \psi(x_0, x_1, \dots, x_n)$ и для любого элемента d_0 , $d_0 \in D_I$ имеет место $I \models \psi(x_0, \dots, x_n)[d_1, \dots, d_n]$.

Формула выполнима в интерпретации, если существует хотя бы один набор элементов интерпретации, на котором формула выполнима. Формула истинна в интерпретации, если она выполнима на любом наборе элементов интерпретации.

Формула выполнима, если существует интерпретация, в которой она выполнима.

Формула общезначима, если она истинна в любой интерпретации. Формула без свободных переменных называется замкнутой формулой (предложением).

Пусть Γ — некоторое множество замкнутых формул, $\Gamma \subseteq CForm$. Тогда каждая интерпретация I , в которой выполняются все формулы множества Γ , называется моделью для множества Γ .

Пусть Γ — некоторое множество замкнутых формул, и ψ — замкнутая формула. Формула ψ называется логическим следствием множества предложений (базы знаний) Γ , если каждая модель для множества формул Γ является моделью для формулы ψ , т. е. для любой интерпретации I верно $I \models \Gamma \iff I \models \psi$.

Запись $I \models \psi$ означает, что ψ — логическое следствие Γ .

Теорема о логическом следстве. Пусть $\Gamma = \{\psi_1, \dots, \psi_n\} \subseteq CForm$, $\phi \in CForm$. Тогда $\Gamma \models \phi \iff \models \psi_1 \wedge \dots \wedge \psi_n \rightarrow \phi$.

$\Delta \models \Gamma$ — пусть $\Gamma \models \phi$. Пусть $\Gamma = \psi_1 \wedge \dots \wedge \psi_n \rightarrow \phi$. Означает, что $\psi_1 \wedge \dots \wedge \psi_n$ — общезначимая формула.

$\Gamma \models \phi \iff \models \psi_1 \wedge \dots \wedge \psi_n \rightarrow \phi$. Так как Γ — модель для Γ , приходим к заключению $\Gamma \models \phi$. ■

Общезначимые формулы — это каналы причинно-следственной связи, по которым передаются знания, представленные в виде логических формул, преобразуются при этом из одной формы в другую.

Практически важно

DOP 8 Язык ассемблера как машиннозависимый язык низкого уровня. Организация ассемблерной программы, секции кода и данных (на примере ассемблера nasm или tasm). Основные этапы подготовки к счёту ассемблерной программы: трансляция, редактирование внешних связей (компоновка), загрузка.

Язык ассемблера

Как правило, вычислительная система состоит из следующих основных компонентов: центральный процессор, оперативная память и внешние устройства.

Программа, предназначенная к выполнению, записывается в оперативную память в виде последовательности машинных инструкций (команд), т.е. цифровых кодов, обозначающих те или иные операции.

Ассемблер – это программа, принимающая на вход текст, содержащий условные обозначения машинных программ, удобные для человека, и переводящий эти обозначения в последовательность соответствующих машинных команд понятного процессору. Язык таких условных обозначений называется **языком ассемблера**.

Программирование на языке ассемблера отличается от программирования на языках высокого уровня. В последних мы задаем лишь указания, а компилятор (программа, принимающая на вход программу) на языке высокого уровня и выдающая эквивалентный машинный код) во всем сам определяет, какими ресурсами (регистрами и ячейками памяти) воспользоваться для хранения промежуточных результатов, как алгоритм применять для решения какой-нибудь нетривиальной ситуации и т.д. В отличие от этого на языке ассемблера мы однозначно и недвусмысленно указываем, из каких машинных команд будет состоять наша программа, в этом понимании ассемблер не имеет практически никакой свободы.

Структура ассемблерной программы

Операционная система может установить пользовательской программе разные возможности по доступу к различным областям памяти. Область памяти может быть доступна для чтения, записи и исполнения. В связи с этим, в современных языках ассемблера существует разделение виртуального адресного пространства на различные области памяти, так называемые секции. Эти секции определяются программистом в ходе создания программы. В результате перевода (трансляции) текста программы каждая секция будет превращена в последовательность байт.

При запуске программы каждая такая последовательность байт будет загружена в оперативную память и размещена в выделенных для нее ячейках памяти. В простых программах набор секций, как правило, ограничен тремя (не считая служебных):

- .text – секция кода.
- .data – секция статических инициализированных данных.
- .bss – секция статических неинициализированных данных, значения которых обнуляются операционной системой перед запуском программы.

Основные этапы подготовки к счёту ассемблерной программы

1. **Трансляция** – это перевод программы на языке ассемблера (в меморииках) в машинные коды. После трансляции получается объективный файл.

2. **Компоновка** (связывание). На входе компоновщика один или несколько объективных файлов, на выходе – программа, готовая к исполнению (исполняемый файл). Компоновщик объединяет несколько файлов в один, пересчитывает адреса, связывает вызовы функций из разных файлов, для вызовов библиотечных функций (при статическом связывании) – добавляет их код в исполняемый файл и также связывает вызовы.

3. Во время загрузки программы секции из исполняемого файла загружаются в память и управление передается в точку входа программы. Также при загрузке происходит связывание с динамическими библиотеками.

Пример ассемблерной программы (язык nasm)

Программа, которая печатает 2 числа на экран:

```
; Equivalent C code
; int main()
; {
;     int a=5;
;     printf("a=%d, eax=%d\n", a, a+2);
;     return 0;
; }

    extern printf      ; the C function, to be called
    SECTION .data      ; Data section, initialized variables
a: dd 5              ; int a=5;
    int a=5;
    printf("a=%d, eax=%d\n", 10, 0 ; The printf format, "\n", 0',
fmt:db "a=%d, eax=%d\n", 10, 0 ; The printf format, "\n", 0',

SECTION .text        ; Code section.
    global main          ; the standard gcc entry point
main:   ; the program label for the entry point
    push ebp
    mov ebp,esp

    mov eax, [a]          ; put a from store into register
    add eax, 2
    push eax
    push dword [a]
    push dword fmt
    call printf
    add esp, 12
    mov esp, ebp
    pop esp
    mov eax,0
    ret

    ; normal, no error, return value
```

[Лекции асм 1 потока]

DOP 6 Теорема Редфилда-Пойа (без доказательства). Примеры применения.

Пусть даны:

- группа $\langle G, \circ, e \rangle$, $|G| = n$
- множество $T, |T| = N > 0$
- $B_{ij}(T)$ – множество всех перестановок элементов T (биекций на T). $B_{ij}(T)$ – симметрическая группа множества T : $S_T = \langle B_{ij}(T), \cdot, 1_T \rangle$.

Действием α группы G на множестве T называется гомоморфизм из группы G в группу S_T .

Отношением эквивалентности \sim_g на T для перестановки g называется $t \sim_g t' \Leftrightarrow \exists k \in \mathbb{Z}: g^k(t) = t'$.

g-циклами называются смежные классы эквивалентности \sim_g . Элементы этих классов образуют циклы: $t \xrightarrow{g} t' \xrightarrow{g} \dots \xrightarrow{g} t$.

Типом перестановки называется $Type(g) = \langle v_1, v_2, \dots, v_N \rangle$, где v_1, v_2, \dots, v_N – количество циклов длины 1, 2, …, N .

*Число всех g -циклов равняется $C(g) = \sum_{k=1}^N v_k(g)$.

Пример: $T = \{1, \dots, 10\}$ и

$$g = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 9 & 6 & 1 & 8 & 5 & 2 & 7 & 10 & 3 & 4 \end{pmatrix} = (1, 9, 3)(2, 6)(4, 8, 10)(5, 7)$$

Тогда $Type(g) = (2, 1, 2, 0, 0, 0, 0, 0, 0, 0)$, $C(g) = 5$, $N = 10$.

Весом $w(g)$ перестановки $g \in G$ называется $w(g) = x_1^{v_1} \dots x_N^{v_N}$, где $\langle v_1, v_2, \dots, v_N \rangle = Type(g)$.

Цикловым индексом $Z(G \underset{\alpha}{;} T, x_1, \dots, x_N)$ действия группы G на множестве T называют средний вес подстановок в группе: $Z(G \underset{\alpha}{;} T, x_1, \dots, x_N) = \frac{1}{|G|} \sum_{g \in G} w(g) = \frac{1}{|G|} \sum_{g \in G} x_1^{v_1} \dots x_N^{v_N}$.

Теорема Пойа

Пусть задано множество T , группа G и действие $\underset{\alpha}{\circ}$ на T .

1. Принимем каждому элементу из T одно из r значений (неформально: покрасим в один из r цветов). Всего, очевидно, имеется r^n раскрасок.
2. Не будем различать раскраски, если элементы t и t' есть $g(t) = g(t')$ раскрашены одинаково.

Тогда число незэквивалентных раскрасок равно числу классов эквивалентности и вычисляется по формуле $C(G \underset{\alpha}{;} T) = Z(G \underset{\alpha}{;} T, x_1, \dots, x_N) \Big|_{x_1 = \dots = x_N = r}$.

Пример. Задача о квадратах 2×2 .

Условие: Сколькими способами можно раскрасить доску 2×2 в r цветов, если раскраски, переходящие друг в друга при вращении квадрата, считаются одинаковыми?

Решение:

Рассмотрим группу вращений квадрата $\mathbb{Z}_4 = \{e, t, t^2, t^3\}$, где e – вращение квадрата на 0, а t – вращение на 90 (очевидно, что $t^4 = e$).

Рассмотрим квадрат $A = \begin{bmatrix} 1 & 2 \\ 4 & 3 \end{bmatrix}$ (это не матрица, а квадрат в котором 4 ячейки под номерами 1, 2, 3 и 4).

$$A = \begin{bmatrix} 1 & 2 \\ 4 & 3 \end{bmatrix}, tA = \begin{bmatrix} 4 & 1 \\ 3 & 2 \end{bmatrix}, t^2A = \begin{bmatrix} 3 & 4 \\ 2 & 1 \end{bmatrix}, t^3A = \begin{bmatrix} 2 & 3 \\ 1 & 4 \end{bmatrix}$$

Таким образом

$$e = (1)(2)(3)(4), Type(e) = \langle 4, 0, 0, 0 \rangle, w(e) = x_1^4$$

$$t = (1432), Type(t) = \langle 0, 0, 0, 1 \rangle, w(e) = x_4^1$$

$$t^2 = (13)(24), Type(t^2) = \langle 0, 2, 0, 0 \rangle, w(e) = x_2^2$$

$$t^3 = (1234), Type(t^3) = \langle 0, 0, 0, 1 \rangle, w(e) = x_1^4$$

$$\text{Итого, цикловый индекс равен } Z = \frac{1}{|G|} (x_1^4 + 2x_4^1 + x_2^2), \text{ подставляя}$$

$$|G| = 4 \text{ и } x_1 = x_2 = x_4 = r, Z(r) = \frac{r^4 + 2r + r^2}{4}.$$

Ответ: $Z(r) = \frac{r^4 + 2r^2}{4}$.

Пример. Задача об ожерельях $N = 5, r = 3$.

Условие:

Сколько разных ожерелий можно составить из 5 бусин 3 цветов?

Решение:

Рассмотрим группу вращений таких ожерелий $\mathbb{Z}_5 = \{e, t, t^2, t^3, t^4\}$, где e – тождественная подстановка, а t – вращение на одну бусину (очевидно, что $t^5 = e$).

$$e = (1)(2)(3)(4)(5), Type(e) = \langle 5, 0, 0, 0, 0 \rangle, w(e) = x_1^5$$

$$t = (12345), Type(t) = \langle 0, 0, 0, 0, 1 \rangle, w(e) = x_5^1$$

$$t^2 = (13524), Type(t^2) = \langle 0, 0, 0, 0, 1 \rangle, w(e) = x_5^2$$

$$t^3 = (14253), Type(t^3) = \langle 0, 0, 0, 0, 1 \rangle, w(e) = x_5^3$$

$$t^4 = (15432), Type(t^4) = \langle 0, 0, 0, 0, 1 \rangle, w(e) = x_5^4$$

Итого, цикловый индекс равен $Z = \frac{1}{|G|} (x_1^5 + 4x_5^1 + 4x_5^2 + 4x_5^3 + 4x_5^4)$, подставляя $|G| = 5$ и

$$x_1 = x_5 = 3, Z(3) = \frac{1}{5} (3^5 + 4 \cdot 3) = 51.$$

Ответ: 51.

[Гуроу. Методические материалы по курсу прикладная алгебра: алгебраические основы кодирования, теории перечислений и шифрования, page 132-161]

DOP 4 Логическое программирование. Декларативная семантика и операционная семантика; соотношение между ними. Стандартная стратегия выполнения логических программ.

$\Box \sigma = \langle Const, Func, Pred \rangle$ – некоторая сигнатура, в которой определяются термины и атомы.

«заголовок» := «атом» | «тело» | «заголовок» \leftarrow «тело»;

«правило» := «заголовок» | «факт»;

«утверждение» := «правило» | «факт»;

«программа» := «пусто» | «утверждение» | «программа»;

«запрос» := \square | ? «тело»;

$\Box G = ?C_1, C_2, \dots, C_m$ – запрос. Тогда

- атомы C_1, C_2, \dots, C_m называются подцелеми запроса G ,

- переменные множества называются целевыми переменными,

- запрос \square называется пустым запросом,

- запросы будем также называть целевыми утверждениями.

Полисемантичность – одна и та же логическая программа имеет две интерпретации (смысли), две смысли.

Программисту важно понимать, что вычисляет программа. Такое понимание программы называется **декларативной** семантикой программы.

Декларативная семантика

Правило $A_0 \leftarrow A_1, A_2, \dots, A_n$:

Если выполнены условия

Чтобы решить задачу A_0 , достаточно решить задачи

A_1, A_2, \dots, A_n , то справедливо утверждение A_0 .

Факт A_0 :

Утверждение A_0 считается

Задача A_0 объявляется реальной.

Запрос $?C_1, C_2, \dots, C_m$

При каких значениях целевых переменных будут верны все отношения C_1, C_2, \dots, C_m .

Решить список задач

все C_1, C_2, \dots, C_m .

Логическая программа

Декларативная семантика

Правильное выражение

DOP 9 Операционные системы. Управление оперативной памятью в вычислительной системе. Алгоритмы методы организации и управления страницей оперативной памяти.

Операционная система — комплекс программ, используемых для управления ресурсами компьютера и предоставления интерфейса пользователя. В понятие управление ресурсами входит **выделение** ресурсов для программ (например, памяти и процессорного времени), **защита** от доступа программ к ресурсам, которым они не владеют, а также **австралирование** от оборудования, например, предоставление общего интерфейса для похожих типов устройств (общий файловый интерфейс для всех дисков) или реализация виртуальных ресурсов (увеличение эффективного объема памяти за счет файла подкачки).

Основные функции ОС:

- Управление процессами — проблемы формирования процессов, поддержание жизненного цикла процесса, проблемы организации взаимодействия процессов, работа процессов с ресурсами.
- Управление оперативной памятью — выбор стратегии организации ОП (реализация поддержки аппаратной виртуальной памяти), защиты ОП от несанкционированного доступа, задачи корректности работы процессов с выделенной ему ОП.
- Планирование — распределение времени центрального процессора, организация и обработка очередей обмена (речется проблема конкуренции доступа к устройству), обработка прерываний.

- Управление внешними устройствами и файловой системой. Файловая система рассматривается как виртуальное устройство. Подразумевается способ обмена в частности ввода и вывода данных в ВС, которая подразумевает использование буфера для временного хранения данных.

- Обеспечение сетевого взаимодействия — ОС должна обеспечивать функционирование и реализацию сетевых протоколов.

- Обеспечение безопасности — устойчивость системы к возможным атакам, анализ функционирования системы и выявление попыток вторжения в систему.

См. типы ос + базовая архитектура — основная часть билет № 18

Оперативное запоминающее устройство (ОЗУ) — устройства для хранения данных, в котором находится исполнимые в данный момент программы. Управление оперативной памятью включает в себя решение четырех задач:

- Контроль использования ресурсов - учет состояния каждой доступной в системе единицы памяти.
- Стратегия распределения памяти - какому процессу, в течение какого времени и в каком объеме выделять соответствующий ресурс.

- Конкретное выделение ресурса тому или иному потребителю - для представляемого ресурса идет корректировка системных данных, а затем выдача его потребителю.

- Освобождение памяти - окончательное освобождение памяти, происходящее в случае завершения процесса + освобождение памяти - задача принятия решения в случае, когда встает потребность высвободить физическую память из-под какого-то процесса за счет откатаивания во внешнюю память, чтобы на освободившемся пространстве поместить данные другого процесса.

Существует несколько моделей по управлению ОП:

- Одиночное непрерывное распределение — все адресное пространство разделяется на две компоненты: в одной части располагается и функционирует операционная система, вторая часть отводится для выполнения прикладных процессов. Для разделения используется регистр границы - если получаемый исполнительный адрес оказывается меньше значения этого регистра, то это адрес пространства ОС, иначе — пространства процесса.

- Распределение непрерываемыми разделами — все адресное пространство делится на две части: одна часть отводится под ОС, оставшаяся — делится на N частей (разделов) и определяется под работу прикладных процессов. Два варианта распределения процессов: общая очередь и очередь каждому разделу. Необходимо использовать два регистра границы - регистры, отвечающие за начало области и ее конец или механизм ключей защиты, которые могут находиться в слове состояния процесса и в слове состояния процессора. При обращении к памяти совпадали — доступ разрешен.

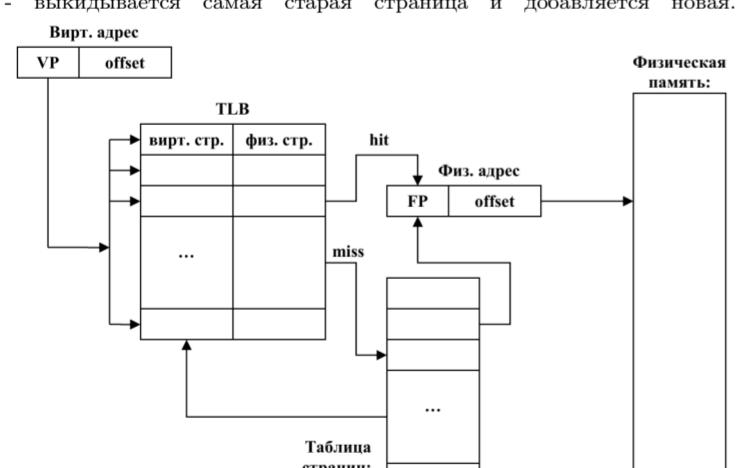
- Распределение перемещаемыми разделами. Исполняемый код процесса может перемещаться по оперативной памяти. Локальная компрессия — система для выравнивания памяти передавать небольшое количество процессов. Глобальная компрессия — система приостанавливает работу всех процессов и начинает их перемещать, например, к начальному адресу ОЗУ, так, вся свободная память — в конце ОЗУ. Используются аппаратные средства защиты памяти - регистры границ или ключи защиты, для перемещения используют регистр базы. Избавляет от фрагментации.

- Стацическое распределение. Все адресное пространство — совокупность блоков фиксированного размера, которые называются **страницами**. **Виртуальное адресное пространство** — пространство с адресами, от которого опирается программа. **Физическое пространство** — пространство, которое есть в наличии в компьютере.

- Сегментное распределение — каждый процесс — совокупность сегментов определенного размера: сегмент кода, сегмент стат. данных, сегмент стека. Используется некоторая таблица, которой хранится информация о каждом сегменте - его номер, размер и тд. Виртуальный адрес интерпретируется, как номер сегмента и величина смещения в нем.

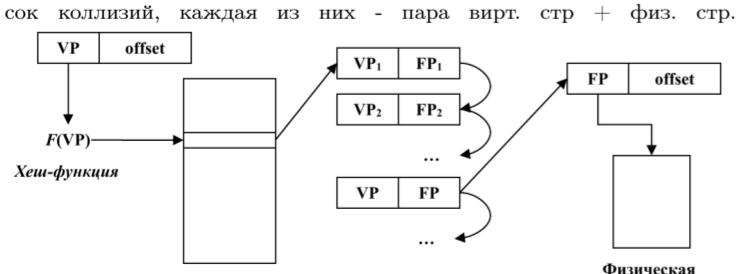
Страницное распределение. Устанавливается соответствие между виртуальными и физическими страницами. Для преобразования виртуального адреса в физический используется таблица страниц. Типовая структура записи таблицы страниц содержит информацию о номере физической страницы, а также совокупность атрибутов, например, флаг модификации содержимого страницы, присутствие/отсутствие страницы и другие.

• Использование TLB-таблицы. Виртуальный адрес = номер виртуальной страницы + адрес смещения в ней. Если промах - выделяется самая старая страница и добавляется новая.

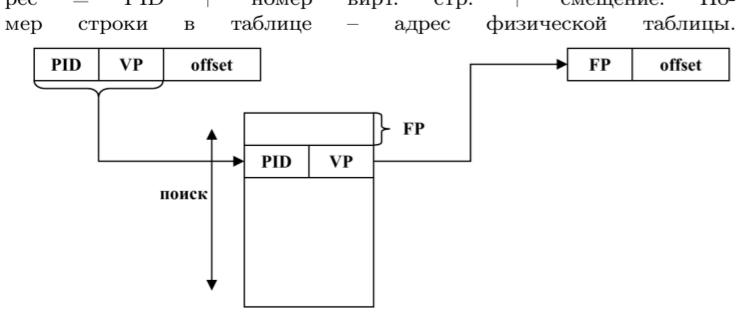


• Иерархическая организация таблиц страниц. Информация о странице представляется в виде совокупности номеров, используя которые, можно получить номер соответствующей физической страницы.

• Хеширование. Извлекается номер виртуальной страницы, который подается в хеш-функцию, отображающей значение на аппаратную таблицу фиксированного размера. Каждая ее запись — список коллизий, каждая из них — пара вирт. стр + физ. стр.



• Инвертированные таблицы страниц. Виртуальный адрес = PID + номер вирт. стр. + смещение. Номер строки в таблице — адрес физической таблицы.



[Пупкин-Залупкин, *Напиши источник!*, page 69-96]

DOP 11 Закон Амдала, его следствия. Граф алгоритма. Критический путь графа алгоритма, ярусно-параллельная форма графа алгоритма. Этапы решения задач на параллельных вычислительных системах.

Ускорение, получаемое при использовании параллельного алгоритма для p процессоров выражается: $S = \frac{T_1}{T_p}$, где T_1 — время выполнения задачи на одном процессоре, T_p — время параллельного выполнения задачи на p процессорах.

Закон Амдала: $S \leq \frac{1}{f + \frac{1-f}{p}}$, где f — доля операций, которые обязаны быть выполнены последовательно ($0 \leq f \leq 1$), p — число процессоров.

Следствие 1. Для того чтобы ускорить программу в q раз, необходимо не менее, чем в q раз, ускорить не менее, чем $\left(1 - \frac{1}{q}\right)$ -ю часть программы. Это необходимо, но не достаточное условие.

Следствие 2. (при большом числе процессоров): $S \approx \frac{1}{f}$.

Граф алгоритма

Будем представлять программы с помощью графов.

Граф алгоритма — ориентированный граф, состоящий из вершин, соответствующих операциям алгоритма, и направленных дуг, соответствующих передаче данных (результаты одних операций передаются в качестве аргументов другим операциям) между ними. Не следует путать его с графиком управления программы и тем более с её блок-схемой. Вершины: процедуры, циклы, линейные участки, операторы, итерации циклов, срабатывания операторов.

Операции — одна вершина для каждой операции. Срабатывания операторов — столько вершин, сколько раз каждый оператор сработал.

Дуги отражают связь (отношение) между вершинами.

Особенностью графа алгоритма являются:

- его ацикличность, нет кратных дуг;
- невозможность, в общем случае, его описания простым перечислением, в силу того, что его составляющие могут зависеть от внешних параметров решаемой им задачи (например, алгоритм, реализующий метод Гаусса — от размера матрицы)



Выделяют два типа отношений: операционное и информационное.

Операционное отношение: две вершины А и В соединяются направлена дугой $A \Rightarrow B$ тогда и только тогда, когда вершина В может быть выполнена сразу после вершины А.

Информационное отношение: две вершины А и В соединяются направленной дугой $A \Rightarrow B$ тогда и только тогда, когда вершина В используется в качестве аргумента некоторое значение, полученное в вершине А.

Операционный граф: вершины — операции, дуги — операционные отношения.

Граф операционной истории программы: вершины — срабатывания операторов, дуги — операционные отношения.

Информационный график: вершины — операции, дуги — информационные отношения.

Граф информационной истории программы: вершины — срабатывания операторов, дуги — информационные отношения.

Независимо от способа построения ориентированного графа, те его вершины, которые не имеют ни одной входящей или выходящей дуги, будем называть соответственно **входными** или **выходными** вершинами.

Информационная история программы.
Вершины: срабатывания операторов
Дуги: информационное отношение

```
for (i = 0; i < n; ++i) {
    A[i] = A[i - 1] + 2;
    B[i] = B[i] + A[i];
}
```

Критический путь графа алгоритма — это длина самого длинного пути от <входа> графа к его <выходу> (от первого оператора до последнему).

Ярусно-параллельная форма графа алгоритма позволяет описать ресурс параллелизма программ и алгоритмов.

- Начальная вершина каждой дуги расположена на ярусе с номером меньшим, чем номер яруса конечной вершины.
- Между вершинами, расположенными на одном ярусе, не может быть дуг.

Высота ЯПФ — это число ярусов
Ширина яруса — число вершин, расположенных на ярусе

Ярусно-параллельная форма графа алгоритма — это сложность параллельной реализации алгоритма/программы. ЯПФ определяется неоднозначно.

Ярусно-параллельная форма называется **канонической**, если у любой вершины, кроме вершины первого яруса, есть входная дуга, идущая с предыдущего яруса. Высота канонической ЯПФ = длине критического пути + 1.

Этапы решения задач на параллельных вычислительных системах Это точно то о чём спрашивается! Решение задачи на параллельной вычислительной системе будет эффективным только в том случае, если на всем пути от Задачи до Компьютера не возникнет ни одного <узкого места>. Центральная проблема: отображение программ и алгоритмов на архитектуру параллельных вычислительных систем (Co-Design).

Весь процесс решения некоторой задачи на параллельной ВС можно разбить на следующие этапы:

1. Формулировка задачи.

2. Составление модели исследуемого в задаче объекта.

3. Определение метода решения задачи для получения необходимой результирующей информации на основании используемой модели.

4. Разработка алгоритма решения задачи на основе используемого метода и модели исследуемого в задаче объекта.

5. Выбор технологии программирования.

6. Разработка программы для соответствующей параллельной ВС на основе имеющегося алгоритма и получение результирующей информации после выполнения программы.

[Пупкин-Залупкин, *Напиши источник!*, page 69-96]

DOP 13 Функции FIRST и FOLLOW. LL(1)-грамматики. Конструирование таблицы предсказывающего анализа.

LL(1)-анализатор — находящий алгоритм синтаксического разбора. Пифа 1 говорит, что для определения пути разбора нужна всего одна лексема.

LL(1) Используется для разбора кода в ряде языков программирования, таких, как Pascal и Python (до 3.8). Очень быстр в исполнении и имеет характерное сообщение об ошибке вида «ожидался такой-то символ».

Определения

- При построении таблицы предсказывающего анализа нам потребуются две функции FIRST и FOLLOW.

- Пусть $G = (N, T, P, S)$ — КС-грамматика. Для α — произвольной цепочки, состоящей из символов грамматики, — определим $FIRST(\alpha)$ как множество терминалов, с которых начинаются строки, выводимые из α . Если $\alpha \Rightarrow^* e$, то e также принадлежит $FIRST(\alpha)$.

- Определим $FOLLOW(A)$ для нетерминала A как множество терминалов a , которые могут появиться непосредственно справа от A в некоторой сенцентиальной форме грамматики, то есть множество терминалов a таких, что существует вывод вида $S \Rightarrow^* \alpha A a \beta$ для некоторых $\alpha, \beta \in (N \cup T)^*$. Заметим, что между A и a в процессе вывода могут находиться нетерминальные символы, из которых выводится e . Если A может быть самым правым символом некоторой сенцентиальной формы, то e также принадлежит $FOLLOW(A)$.

• Алгоритм имитации отжига основан на аналогии с физическим процессом отжига металлов. На каждом шаге с вероятностью, зависящей от текущей «температуры» T , допускается переход в состояние с худшим значением целевой функции:

DOP 15 Алгоритмы имитации отжига. Проблема возможности потери лучшего решения. Способы спараллелизации алгоритма имитации отжига.

Алгоритм имитации отжига основан на аналогии с физическим процессом отжига металлов. На каждом шаге с вероятностью, зависящей от текущей «температуры» T , допускается переход в состояние с худшим значением целевой функции:

$$P(X^k \rightarrow X^{k+1}) = \begin{cases} 1, & \Delta F \leq 0 \\ \exp\left(-\frac{\Delta F}{T}\right), & \Delta F > 0 \end{cases}$$

где $\Delta F = F(X') - F(X)$.

Законы снижения температуры

Температура T снижается в соответствии с выбранным законом, что влияет на баланс между исследованием пространства решений и эксплуатацией текущего минимума. Основные подходы:

- **Закон Больцмана:**

$$T(i) = \frac{T_0}{\ln(1+i)},$$

где i — номер итерации. Медленное снижение, подходит для задач с большим пространством поиска.

- **Закон Коши:**

$$T(i) = \frac{T_0}{1+i}.$$

Более быстрое снижение температуры, уменьшает время вычислений.

DOP 16 Основные понятия криптографии. Односторонняя функция с секретом. Протокол Диффи-Хеллмана выработки общего секретного ключа по открытому каналу связи.

Криптография — наука о способах преобразования (шифрования) информации с целью её защиты от незаконных пользователей, обеспечения целостности и реализации методов проверки подлинности.

Открытый текст — сообщение, подлежащее шифрованию.

Шифртекст (криптограмма) — результат шифрования открытого текста.

Шифр — семейство обратимых отображений множества последовательностей открытых текстов во множестве последовательностей шифртекстов.

Ключ — параметр, определяющий выбор конкретного отображения.

Зашифрование — процесс преобразования открытого текста в шифртекст с помощью шифра и ключа к данному тексту.

Расшифрование — процесс, обратный к зашифрованию, реализуемый при известном значении ключа.

Дешифрование — процесс раскрытия криптограммы без знания секретного ключа.

Односторонняя функция — обратимая функция: $X \rightarrow Y$, обладающая свойствами:

- Э полиномиальный алгоритм вычисления значений $f(x)$.
- $\frac{d}{dx}$ полиномиального алгоритма обращения функции f .

Односторонняя функция с секретом (с лазеркой) — функция $f_k(x) : X \rightarrow Y$ зависящая от параметра k , называемым секретным ключом.

1. Вычисление значения $f_k(x)$ относительно несложно и при этом не требуется значение параметра k .
2. Вычисление значения $f_k^{-1}(y)$ для всех $y \in Y$ при известном k относительно несложно.
3. Потом для всех k и $y \in Y$, нахождение $f_k^{-1}(y)$ вычислительно несущественно без знания k .

Один из примеров, претендующих на то, чтобы являться односторонней функцией с лазеркой — функция $f(x) = x^m \pmod n$, n и m известны. Вычисление $f(x) = y$ производится методом быстрого возведения в степень, а эффективный алгоритм обратного преобразования $f^{-1}(y)$, то есть вычисление корня m -ой степени по модулю n , требует примарного разложения n [типо разложение на простые числа]. Эта информация может считаться лазеркой.

Простые поля Галуа — поле классов вычетов по модулю простого числа. Пример: $Z_3 = \{n \pmod 3\} = \{0, 1, 2\}$

Порождающие элементы мультипликативной группы поля [м.г. — типо все элементы поля без нуля] называют его **прimitивными элементами**.

Пример: числа 2, 6, 7, 8 — примитивные элементы поля F_{11}

Проверка 2: $2^k \pmod {11} = 2^k \pmod 11$

$2^5 \pmod {11} = 2^5 \pmod 11 = 32 \pmod 11 = 10 \pmod 11 = 10 \neq 1 \Rightarrow 2$ — примитивный.

$3^5 \pmod {11} = 3^5 \pmod 11 = 243 \pmod 11 = 1 \Rightarrow 3$ — не примитивный.

Выработка общего секретного ключа по открытому каналу связи (протокол Диффи-Хеллмана)

Алиса (А) и Боб (Б) обмениваются сообщениями по открытому каналу. Для обеспечения секреции нужен общий секретный ключ.

1. А и В выбирают простое число p (число р нужно выбирать очень большим, чтобы было очень сложно "взломать" подбором - аналитически не решается) и в (простом) поле Галуа $GF(p)$ некоторый примитивный элемент α . Данные значения не являются секретом.
2. А и В независимо друг от друга выбирают по одному случайному натуральному числу x и y соответственно, которые держат в секрете.

3. Далее они вычисляют, соответственно, $X = \alpha^x \pmod p$, $Y = \alpha^y \pmod p$. X и Y передаются друг другу по открытому каналу.
4. А вычисляет: $K = Y^x \pmod p$. В вычисляет: $K = X^y \pmod p$.

5. Таким образом, А и В получают общий секретный ключ $K = \alpha^{xy} \pmod p$, который в дальнейшем используется в алгоритмах симметричного шифрования.

Пассивный злоумышленник, перехвативший, но не изменяющий сообщений не может определить ключ K : его определение связано с решением одного из ур-шага 3, а это вычислительно трудная задача дискретного логарифмирования.

Протокол Диффи-Хеллмана подвержен атаке "человек по середине": если к каналу связи имеет доступ активный злоумышленник, вырабатывает два ключа — общий с А и общий с В, он может представиться Алисе Бобом, а Бобу — Алисой.

[С. И. Гуров, Запись лекций по курсу прикладной алгебра. Алгебраические основы кодирования и шифрования, page 139-151]

DOP 14 Понятие имитационной модели. Примеры средств имитационного моделирования. Типовая архитектура средств имитационного моделирования (OMNeT++, NS3).

Понятие имитационной модели

Имитационная модель — это алгоритмическая математическая модель, отражающая поведение исследуемого объекта во времени при задании внешних воздействий. Она воспроизводит процесс функционирования системы, имитируя элементарные явления с сохранением их логической структуры и временной последовательности.

Основные особенности:

- **Алгоритмическая основа:** описывает поведение объекта через алгоритмы, учитывающие изменения состояний во времени.
- **Динамическое представление:** моделирует процессы в реальном или условном времени, сохраняя логику взаимодействий.
- **Универсальность:** применима для широкого круга задач, но результаты зависят от конкретных входных данных.
- **Методы продвижения времени:** постоянный шаг (непрерывные/потоковые модели), дискретно-событийный подход, гибридные методы.
- **Интеграция предметной области:** требует поддержки специфических понятий и логики системы в инструментах моделирования.

Этапы построения имитационной модели

Основные этапы:

• Анализ требований и проектирование:

- Определение целей моделирования (например, оценка производительности системы, анализ загрузки ресурсов).
- Построение концептуальной модели:
 - * Формализация структуры системы (компоненты, связи, состояния).
 - * Описание алгоритмов функционирования (взаимодействие элементов, обработка событий).
 - * Упрощение и абстракция (исключение несущественных деталей).
- Проверка адекватности концептуальной модели (экспертная оценка, сравнение с реальным объектом).

• Реализация модели:

- Выбор инструментов: язык программирования (C++, Python), библиотеки (SimpleSim), среды моделирования.
- Программирование компонентов:
 - * Реализация логики событий (например, обработка запросов сервером).
 - * Моделирование времени (дискретно-событийный подход, постоянный шаг).
- Отладка и верификация (проверка корректности кода, соответствия концептуальной модели).

• Экспериментирование и анализ:

- Планирование экспериментов:
 - * Выбор варьируемых параметров (интенсивность запросов, количество серверов).
 - * Определение метрик (загруженность сервера, длина очереди).
- Прогон модели:
 - * Запуск с разными наборами входных данных.
 - * Сбор статистики (например, временные диаграммы работы системы).
- Анализ результатов:
 - * Интерпретация данных (выявление «узких мест», оптимизация параметров).
 - * Формулирование выводов (рекомендации по настройке системы).

Пример из лекции: Модель сервера с клиентами. На этапе реализации использован событийный подход с классами Client, Server и календарём событий. В экспериментах анализировались максимальная длина очереди и загруженность сервера.

Примеры средств имитационного моделирования

• OMNeT++:

- Среда для дискретно-событийного моделирования с открытым исходным кодом.
- Использует язык NED для описания иерархии модулей (простые/ составные) и связей между ними.
- Поведение компонентов реализуется на C++ через обработку сообщений (Message) и планирование событий (scheduleAt).
- Примеры применения: моделирование сетей, обработка прерываний, клиент-серверные системы.

• NS-3:

- Преймворк для эмуляции сетевых протоколов (уровень L3 и выше).
- Поддерживает узлы (Node), сетевые интерфейсы (NetDevice) и канали (Channel).
- Сбор данных через трассировку в формате PCAP и журнализование.

• AnyLogic:

- Универсальная платформа, поддерживающая агентное, системно-динамическое и дискретно-событийное моделирование.
- Интеграция с UML-диаграммами состояний для описания параллельных процессов.

Типовая архитектура средств имитационного моделирования

• Общие компоненты:

- Планировщик событий: управляет очередью событий с временными метками (дискретно-событийный подход).
- Моделируемые объекты:

- * В OMNeT++: модули (cSimpleModule) с методами initialize(), handleMessage().
- * В NS-3: узлы, протоколы (TCP/IP), каналы связи.

◦ Средства сбора данных:

- * Логирование (OMNeT++ — .vec/.sca, NS-3 — PCAP).
- * Визуализация через графические интерфейсы (например, Tkenv в OMNeT++)

• Методы организации моделей:

- Последовательные процессы: реализация через потоки с методами main(), signal_event().
- Конечные автоматы:
 - * Состояния с действиями при входе/выходе (например, «Свободен», «Занят»).
 - * Параллельные состояния и синхронизация через триггеры и сторожевые условия.

• Этапы разработки:

- Анализ требований и проектирование концептуальной модели (диаграммы состояний).
- Реализация модели: выбор языка (C++ для OMNeT++, Python для NS-3), отладка.
- Проведение экспериментов: параметризация, анализ результатов.

Характеристика	OMNeT++
Парадигма	Дискретно-событийная + конечные автоматы
Описание структуры	Язык NED
Отладка	Графический интерфейс Tkenv

Примечания

- Для моделирования прерываний в OMNeT++ используются методы scheduleAt() и cancelEvent().
- Примеры применения: моделирование серверов FTP, сокетов, процессов в Linux.
- Гибридные модели сочетают дискретные события и непрерывные процессы для сложных систем.

[Бахмуров, Имитационное моделирование в исследовании и разработке вычислительных систем]

DOP 12 Классификация языков, определяемых конечными автоматами, регулярными выражениями и правилыми грамматиками. Эквивалентность и минимизация конечных автоматов.

Грамматики и регулярные множества и выражения

- **Грамматика** — это четверка $G = (N, T, P, S)$, где N — алфавит нетерминалных символов, T — алфавит терминальных символов, $N \cap T = \emptyset$, P — конечное множество правил вида $\alpha \rightarrow \beta$, где $\alpha \in (N \cup T)^*$ (хотя бы 1), $\beta \in (N \cup T)^*$, $S \in N$ — начальный знак или аксиома грамматики.

- **Отношение выводимости** \Rightarrow — если $\gamma \rightarrow \delta \in P$, то $\alpha\gamma\beta \Rightarrow \alpha\delta\beta$, $\alpha, \beta \in (N \cup T)^*$.

- **Сентенциальная форма** грамматики G — цепочка, выводимая из ее начального символа.

- **Языком**, порождаемым грамматикой G ($L(G)$) называется множество всех ϵ терминальных сентенциальных форм $L(G) = \{w \in T^* | S \Rightarrow^* w\}$.

- Грамматики **эквивалентны**, если они порождают один и тот же язык.

- **Классификация грамматик по Хомскому:**

Тип 0 Любая порождающая грамматика.

- **Тип 1** (Неукараивающая или контексто-зависимая грамматика) Если каждое правило кроме $S \rightarrow e$ имеет вид $\alpha \rightarrow \beta$, где $|\alpha| \leq |\beta|$, и в том случае, когда $S \rightarrow e \in P$, S встречается в правых частях правил.

- **Тип 2** (Контексто-свободная грамматика.) Если каждое правило имеет вид $A \rightarrow \beta$, где $A \in N$, $\beta \in (N \cup T)^*$.

- **Тип 3** (Регулярная праволинейная (или леволинейная) грамматика.) Каждое правило имеет вид либо $A \rightarrow xB$, либо $A \rightarrow x$, где $A \in N$, $x \in T^*$. (для леволинейных поменять x на Bx .)

- **Типом языка** называется максимальный тип грамматики, порождающей этот язык.

- **Регулярное множество** определяется рекурсивно:

1. \emptyset — регулярное множество в алфавите T .
2. $\{e\}$ — регулярное множество в T , где e — пустая цепочка.
3. $\{a\}$ — регулярное множество в T для каждого $a \in T$.
4. Если P и Q — регулярные множества в T , то регулярными являются $P \cup Q$ (объединение), PQ (конкатенация), P^* (терпилация).
5. Ничто другое не является регулярным множеством.

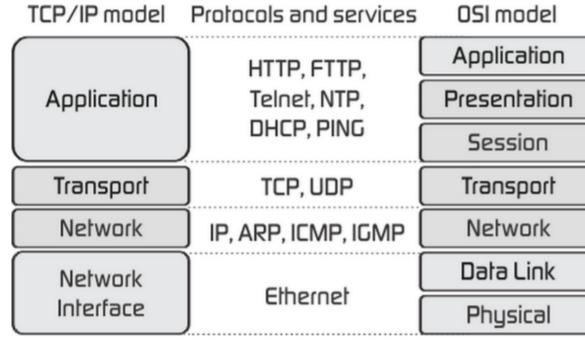
- **Регулярное выражение** определяется рекурсивно:

1. \emptyset — регулярное выражение, обозначающее

DOP 17 Основные принципы построения и архитектура сети Интернет. Алгоритмы и протоколы внешней и внутренней маршрутизации. Явление перегрузки и методы борьбы с ней.

Основные принципы построения сети Интернет

- 1. **Децентрализация управления** — нет единого центра управления для сети Интернет.
- 2. **Выход из строя одного компьютера или участка сети не приводит к неработоспособности всей сети.**
- 3. **Коммутация пакетов** — способ динамического распределения ресурсов сети связи за счёт передачи и коммутации оцифрованной информации в виде частей небольшого размера, так называемых **пакетов**, которые передаются по сети, в общем случае, независимо друг от друга (действиями) либо последовательно друг за другом по виртуальным соединениям. Узел-приёмник из пакетов собирает сообщение. В таких сетях по одной физической линии связи могут обмениваться данными много узлов.
- 4. **Инкапсуляция** — последовательное вложение протокольной единицы (PDU) вышестоящего уровня в протокольную единицу нижележащего уровня. PDU вышестоящего уровня не доступны на нижележащем уровне (нижележащий уровень не понимает структуры данных вышестоящего уровня).
- 5. Каждый уровень имеет строго определенный интерфейс с нижележащим и вышестоящим уровнями и набор сервисов, которые может использовать вышестоящий уровень.



ISO/OSI:

Физический — передача последовательности битов через физическую линию.
Канальный — превращение несовершенной физической среды передачи данных в надежный канал, свободный от ошибок передачи.
Сетевой — построение маршрута между отправителем и получателем.
Транспортный — получение данных с высокосложного уровня, разделение на более мелкие единицы (если требуется), передача на сетевой уровень, обеспечение целостности данных при доставке до адресата.

Сессии — установка сессий между пользователями. Сессия позволяет передавать данные, как это может делать транспортный уровень, и имеет более сложный сервис, полезный в некоторых приложениях. Например, можно осуществить вход в удаленную систему.

Представления — обеспечение решений проблем, связанных с представлением данных, в основном — проблемами синтаксиса и семантики передаваемой информации.

Приложения — обеспечение работы часто используемых приложений, например — передача файлов.

TCP/IP:

Прикладной уровень. На прикладном уровне (Application layer) работают большинство сетевых приложений. Эти программы имеют свои собственные протоколы обмена информацией, например, интернет браузер для протокола HTTP, ftp-клиент для протокола FTP (передача файлов).

Транспортный уровень (как упаковывать?). Протоколы транспортной уровня (Transport Layer) могут решать проблему негарантированной доставки сообщений («дошло ли сообщение до адресата?»), а также гарантировать правильную последовательность прихода данных.

Сетевой (межсетевой) уровень (кому отправлять?). Межсетевой уровень (Network layer) изначально разработан для передачи данных из одной сети в другую. На этом уровне работают маршрутизаторы, которые перенаправляют пакеты в нужную сеть путем расчёта адреса сети по маске сети.

Канальный уровень (как кодировать в среде?). Канальный уровень (англ. Link layer) описывает способ кодирования данных для передачи пакета данных на физическом уровне (то есть специальные последовательности бит, определяющие начало и конец пакета данных, а также обеспечивающие помехоустойчивость).

Алгоритмы и протоколы внешней и внутренней маршрутизации

Алгоритмы маршрутизации применяются для определения наилучшего пути пакетов от источника к приемнику и являются основой протокола маршрутизации. Для формулирования алгоритмов маршрутизации сеть рассматривается как граф. При этом маршрутизаторы являются узлами, а физические линии между маршрутизаторами — ребрами соответствующего графа. Каждой грани графа присваивается определённое число — стоимость, зависящая от физической длины линии, скорости передачи данных по линии или стоимости линии. Основными алгоритмами являются алгоритмы Беллмана-Форда и Дейкстры.

Алгоритм Беллмана-Форда (пример алгоритма по вектору расстояний):

• Пусть каждый маршрутизатор знает стоимость линии к каждому своему непосредственному соседу

• Маршрутизатор R_8 рассчитывает стоимость C_i для достижения каждого известного ему R_i

• Вектор $C_8 = (C_1, C_2, \dots, C_r)$ — вектор расстояния до R_8

• Изначально $C = (\infty, \infty, \dots, \infty)$

1. Каждые T секунд R_8 шлет C_i всем своим соседям

2. Если R_i нашёл более дешёвый путь, то он обновляет C_i у всех своих соседей

3. Вернуться к 1

• Длина вектора C устанавливается администратор

Алгоритм Дейкстры (пример алгоритма по состоянию канала):

1. Определение топологии сети: каждый маршрутизатор передаёт линии всем своим соседям состояния своих линий и строит топологию сети (1. Периодически, 2. Когда изменяется состояние линии)

2. Вычисление по алгоритму Дейкстры: каждый маршрутизатор независимо запускает алгоритм Дейкстры наискратчайшего пути. Каждый маршрутизатор находит соединяющее дерево с минимальной стоимостью до каждого другого маршрутизатора.

Протоколы внутренней маршрутизации: RIP (на основе алгоритма Б-Ф, редко используется), OSPF (на основе алгоритма Д, широко используется).

Протоколы внешней маршрутизации: BGP-4 — алгоритм разработан для того, чтобы решить проблему того, что Интернет представляет из себя смесь различных автономных сетей. Необходимо учитывать ряд условий, связанных с политикой автономной сети, например, пакеты из Министерства обороны не должны проходить через недружественные страны.

Перегрузка — падение производительности в транспортной среде из-за слишком большого числа пакетов. (Если на маршрутизаторе в очереди скапливается много пакетов, то они просто сбрасываются — это неэффективно)

Управление перегрузками — организация потоков в транспортной среде, при которой потоки соответствуют пропускной способности подсети и не превышают ее.



DOP 18 Базисные типы данных в языках программирования. Основные проблемы, связанные с базисными типами и способы их решения в различных языках. Понятие абстрактного типа данных и способы его реализации в современных языках программирования.

Простые типы данных и их свойства

Целевые типы:

- Универсальность (насколько полно учтены машинные типы).
- Наличие (или отсутствие) беззнаковых типов (в Java их нет, в C++, C# есть).
- Представление (размер значения, диапазоны значений).
- Надежность (какие ошибки могут возникать при выполнении операций с целыми значениями — переполнение типа).
- Набор операций (почти во всех языках одинаково, в Java нет беззнакового типа, поэтому есть логический сдвиг).

Вещественные типы:

- $(-1)^S * M * 2^E$, где S — бит знака, M — нормализованнаяmantissa ($0.5 \leq M < 1$), p — порядок.
- Неточные (например, float — точность сложения 2^{-23} , если операнды между 0.5 и 1).

Символьные типы:

- Включают в себя как символы алфавитов естественных языков, так и символы, управляющие работой устройств ввода/вывода, и специальные символы.
- Главной проблемой символьного типа является выбор кодировки. Современное решение — Unicode.

Логические типы: в С отсутствует, в C++ можно преобразовывать к целому, в C#, Java — нет.

Порядковые типы:

- **Перечислимые типы** — перечень именованных значений констант, типы диапазона.

• **Перечислимый тип C++:** числовые значения констант — всегда int , преобразования enum в int — неявно, int в enum — только явно, константы перечислимого типа имеют ту же область действия, что и имя перечислимого типа.

• **Перечислимый тип C#:** константы типа доступны через `<имя типа><имя константы>`, только явные преобразования между enum и int .

• **Java:** аналогично C# и являются классами.

• **Тип диапазона** позволяет ограничить значения целого типа. Есть в Паскале, в C++, C#, Java — нет.

Указательные и ссылочные типы данных:

- **Указатель** абстракция понятия машинного адреса, есть в С, C++ — может привести к хитрым ошибкам. Основные причины использования указателей: передача адресов объектов данных в подпрограммы, работа с объектами из динамической памяти.

• **Ссылка** — абстракция понятия машинного адреса, лишенная недостатков указателя. Ссылки C# и Java отличаются от ссылок C++ тем, что ссылка C++ «навсегда», то есть в течение всего времени жизни ссылки, полностью ассоциирована с объектом. Однако, и ссылки в C++, и ссылки в C# и Java похожи в том смысле, что после установления ассоциации с объектом ссылка идентична самому объекту, поэтому не требуется никакая операция разыменования.

Составные типы и их свойства

1. Одномерные массивы.

Массив — это непрерывная последовательность элементов одного типа. Атрибуты массива: базовый тип (T), тип индекса (I), диапазон индексов (L и R — нижняя и верхняя граница) и связанная с ним характеристика: длина массива. В C# и Java массивы — полноправные объекты.

2. Многомерные массивы.

В большинстве языков рассматриваются как массивы массивов. В Java все многомерные массивы — ступенчатые (то есть внутренние массивы не обязаны иметь одну длину), в C# есть прямоугольный массив (для более эффективного доступа к элементам и возможности обрабатывать массивы, совместимые с моделью данных языков типа С).

3. Динамические строки.

Последовательность символов произвольной длины. Необходимость введения специального типа вместо массива: строки реализуются как неизменяемый объект (главный аргумент), набор операций для строк существенно шире и специфичней набора операций для обычных массивов.

4. Записи (структуры)

— это совокупность объявлений переменных, которые объединены в отдельный объект.

Абстрактные типы данных

Абстрактный тип данных (ATD) — тип, в котором внутренняя структура данных полностью инкапсулирована, то есть тип представлен только множеством операций. Класс является абстрактным типом данных, если открытыми членами являются только методы.

Говорят, что совокупность открытых членов класса составляет **интерфейс** класса.

Реализация:

- Сокрытие членов-данных, доступ через селекторы (геттеры-сеттеры) или их абстракцию — свойство (в C#).

• **Абстрактный класс** — класс, который предназначен исключительно для того, чтобы быть базовым классом.

• **Интерфейс** — класс, состоящий только из абстрактных методов.

[Головин, Материалы по языкам программирования к госэкзамену]

DOP 20 Основные характеристики функциональных языков программирования. Использование понятий функционального программирования (замыкания, анонимные функции) в современных объектно-ориентированных языках.

Свойства функциональных ЯП:

1. Язык динамический — связывания происходят во время выполнения.
2. Нет понятия состояния и присваивания.
3. Главная операция — вызов функции.
4. Главная абстракция — определение функции.
5. Функции — объекты 1 класса, то есть могут быть значениями, вычисляться, передаваться как параметры и возвращаемые значения и т.п.
6. Структуры данных — списки (последовательности).
7. Простая типовая структура.
8. Понятие переменной соответствует математическому смыслу — переменная отождествляется со значением, а не хранит его.

Понятия функционального программирования

- **Замыкание** — это конструкция, которая связывает функцию (функциональное значение) с переменными из области видимости. Про такие переменные говорят, что они «захвачены», их область видимости (scope) не совпадает с областью действия (extent), последняя — шире. Пример:

```
function initAdder(x) {
    function adder(y) {return x + y}
    return adder
}
```

• Анонимная функция (лямбда-функция) — это «чистое» функциональное значение без имени. Его можно передавать как параметр другой функции, возвращать как результат другой функции, в языках с процедурными конструкциями — присваивать.

Использование понятий ФП в современных ОО языках

C#

```
delegate(int x, int y) {return x+y;}
(x,y)=> {return x+y;}
(x,y)=>x+y;
```

Лямбда-выражения (3 строчки) — более общая конструкция, чем лямбда-операторы, могут быть преобразованы в стандартный тип деревьев выражений. Параметры анонимных делегатов типизированы (тип возвращаемого значения выводится из типа выражения в return). А лямбда-конструкции — нетипизированы.

Java

```
(Integer x, Integer y) -> x + y
```

или

```
(Integer x, Integer y) -> return x + y;
```

— лямбда-выражения. Типами параметров лямбда-выражений могут быть только объективные типы, тип возвращаемого значения выводится из возвращаемых выражений.

Пример замыкания:

```
Function<Integer, Integer> initAdder(int x) {
    return (Integer y) -> x + y;
}
```

Пример на Python:

```
square = lambda n: n * n # lambda expression
print(square(4)) # 16
```

[Головин, Материалы по языкам программирования к госэкзамену]

DOP 21 Отказоустойчивость в распределенных системах. Типы отказов. Фиксация контрольных точек

DOP 24 Распределенные файловые системы. Доступ к директориям и файлам. Семантика одновременного доступа к одному файлу нескольких процессов. Кэширование и размножение файлов.

Файловый сервис — это то, что файловая система предоставляет своим клиентам, т.е. интерфейс с файловой системой.

Файловый сервер — это процесс, который реализует файловый сервис.

Интерфейс файлового сервера.

В UNIX и MS-DOS файл — не интерпретируемая последовательность байтов. Большинство PC базируются на использовании среди UNIX и MS-DOS, они используют такие варианты понятия файла. Файл может иметь атрибуты (информация о файле, не являющаяся его частью). Могут ли файлы модифицироваться после создания? Обычно да, но есть системы с неизменяемыми файлами — нет проблем при кэшировании и размножении. Защита обеспечивается теми же механизмами, что и в однопроцессорных ЭВМ — мандатами и списками прав доступа. **Мандат** — своего рода билет, выданный пользователю для каждого файла с указанием прав доступа. Список прав доступа задает для каждого файла список пользователей с их правами. Простейшая — UNIX схема, в которой различают три типа доступа (чтение, запись, выполнение), и три типа пользователей (владелец, члены его группы, и прочие).

Файловый сервис может базироваться на одной из двух моделей — модели загрузки/разгрузки и модели удаленного доступа. В первом случае файл передается между клиентом (памятью или дисками) и сервером целиком, а во втором файл сервер обеспечивает множество операций (открытие, закрытие, чтение и запись части файла, сдвиг указателя, проверку и изменение атрибутов, и т.п.).

Интерфейс сервера директорий.

Обеспечивает операции создания и удаления директорий, именования и переименования файлов, перемещение файлов из одной директории в другую. Определяет алфавит и синтаксис имен. Все распределенные системы используют иерархическую ФС.

Семантика разделения файлов.

UNIX-семантика.

Если за операцией записи следует чтение, то результат определяется последней из предшествующих операций записи. В распределенной системе такой семантики достичь легко только в том случае, когда имеется один файл-сервер, а клиенты не имеют кэшей. При наличии кэшевой семантики нарушается: надо либо сразу все изменения в кэшах отражать в файлах, либо менять семантику разделения файлов.

Еще одна проблема — трудно сохранить семантику общего указателя файла (в UNIX он общий для открытого файла процесса и его дочерних процессов) — для процессов на разных ЭВМ трудно иметь общий указатель.

Неизменяемые файлы — очень радикальный подход к изменению семантики разделения файлов. Только две операции — создать и читать. Можно заменить новым файлом старый, т.е. можно менять директории.

Если один процесс читает файл, а другой его подменяет, то можно позволить первому процессу доработать со старым файлом в то время, как другие процессы могут уже работать с новым.

Семантика сессий.

Изменение открытого файла видны только тому процессу (или машине), который производит эти изменения, а лишь после закрытия файла становятся видны другим процессам (или машинам).

Что происходит, если два процесса одновременно работали с одним файлом — либо результат будет определяться процессом, последним закрывшим файл, либо можно только утверждать, что один из двух вариантов файла станет текущим.

Транзакции. Процесс выдает операцию <НАЧАЛО ТРАНЗАКЦИИ>, сообщая тем самым, что последующие операции должны выполняться без вмешательства других процессов. Затем выдает последовательность чтений и записей, заканчивающиеся операцией <КОНЕЦ ТРАНЗАКЦИИ>.

Если несколько транзакций стартуют в одно и то же время, то система гарантирует, что результат будет таким, каким бы он был в случае последовательного выполнения транзакций (в неопределенном порядке). **Кэширование.**

В системе клиент-сервер с памятью и дисками есть четыре потенциальных места для хранения файлов или их частей.

Во-первых, хранение файлов на дисках сервера. Нет проблем когерентности, так как одна копия файла существует. Главная проблема — эффективность, поскольку для обмена с файлом требуется передача информации в обе стороны и обмен с диском.

Кэширование в памяти сервера. Две проблемы — помешать в кэш файлы целиком или блоки диска, и как осуществлять выталкивание из кэша. Коммуникационные издержки остаются.

Избавиться от коммуникаций позволяет кэширование в машине клиента. Кэширование на диске клиента может не дать преимуществ перед кэшированием в памяти сервера, а сложность повышается значительно. Поэтому рассмотрим подробнее организацию кэширования в памяти клиента, а) кэширование в каждом процессе. (Хорошо, если с файлом активно работает один процесс — многократно открывает и закрывает файл, читает и пишет, например в случае процесса базы данных). б) кэширование в ядре. (Накладные расходы на обращение к ядру). в) кэш-менеджер в виде отдельного процесса. (Ядро облегчается от функций файловой системы, но на пользовательском уровне трудно эффективно использовать память, особенно в случае виртуальной памяти. Возможна фиксация страниц, чтобы избежать обменов с диском).

Оценить выбор того или иного способа можно только при учете характеристики приложений и данных о быстродействии процессоров, памяти, дисков и сети.

Когерентность кэшей.

Алгоритм со сквозной записью.

Необходимость проверки, не устарела ли информация в кэше. Запись вызывает коммуникационные расходы (MS-DOS).

Алгоритм с отложенкой записью.

Через регулярные промежутки времени все модифицированные блоки пишутся в файл. Эффективность выше, но семантика непонятная пользователю (UNIX).

Алгоритм записи в файл при закрытии файла.

Реализует семантику сессий. Не намного хуже случая, когда два процесса на одной ЭВМ открывают файл, читают его, модифицируют в своей памяти и пишут назад в файл.

Алгоритм централизованного управления.

Можно выдержать семантику UNIX, но не эффективно, ненадежно, и плохо масштабируется.

[Курс распределенных систем]

[Бахтин, Презы по распределенным системам, Конспект Лекций.zip/3-4-MPI-Sync.doc: page 8-15]

DOP 22 Синхронизация в распределенных системах. Синхронизация времени. Логические часы. Выборы координатора. Взаимное исключение. Координация процессов.

Основное

Распределенная (компьютерная) система (РС) — совокупность связанных сетью независимых компьютеров, которая представляется пользователю единным компьютером.

Свойства децентризованных алгоритмов:

1. используемая информация распределена среди множества ЭВМ;

2. процессы действуют на основе только локальной информации;

3. отсутствие критического узла, выход из строя которого приводит к крушению алгоритма;

4. отсутствие общего источника глобального времени.

пункты 1–3 о недоступности хранения всей информации необходимой для принятия решения в одном месте.

Синхронизация времени

о **Аппаратные часы** — счетчики временных сигналов и регистр с начальным значением счетчика [из слайдов].

(**Аппаратные часы** — счетчики времени, система содержащая автономный источник питания и регистр, [из вики]).

Отошение «произошло до» (\rightarrow): $a \rightarrow b$ означает, что все процессы согласны, что сплошная произошло событие a , а затем b . Оно очевидно в 2-х случаях:

• оба события (a и b) произошли в одном процессе;

• событие a — отправка сообщения m , событие b — прием m .

Отношение x транзитивно ($a \rightarrow b \rightarrow c$ тогда $a \rightarrow c$).

Если события x и y произошли в разных процессах, не обменивающихся сообщениями, то отношения $x \rightarrow y$ и $y \rightarrow x$ неверны. Такие события x и y называются одновременными.

о **Логические часы** — согласованное (между ЭВМ РС) время (потенциально) не имеющее ничего общего с астрономическим временем.

Введено логическое время C следующим образом: если $a \rightarrow b$, то $C(a) < C(b)$. Логические часы используют следующий алгоритм:

1. Часы C_i увеличивают свое значение с каждым событием в процессе P_i : $C_i = C_i + d$, где $d > 0$ (обычно 1).

2. Если событие a — отправка сообщения m процессом P_i , то в него добавляется временная метка $t_m = C_i(a)$. При получении сообщения m процессом P_j его время корректируется: $C_j = max(C_j, t_m + d)$.

Выбор координатора

Многие распределенные алгоритмы требуют, чтобы один из процессов опорил функции координатора. Рассмотрим алгоритмы выбора координатора.

1. Алгоритм “Задира”:

• Если процесс P обнаружит, что координатор долго не отвечает, то он инициирует выборы.

• P посылает сообщение “ВЫБОРЫ” всем процессам с большим номером у него номерами.

• Если нет ни одного ответа, то P считается победителем и становится координатором.

• Если один из процессов с большим номером ответит, то он берет на себя проведение выборов. Участие процесса P в выборах заканчивается.

• В любой момент процесс может получить сообщение “ВЫБОРЫ” (от процесса с меньшим номером). В этом случае он посылает ответ “OK”, чтобы сообщить, что он жив и берет проведение выборов на себя.

• Победитель извещает всех о своей победе сообщением “КООРДИНАТОР”.

• (Сомнительный пункт: типа зачем, если новый координатор умрет, то управление вернется) Если бывший координатор восстановился после сбоя, то он проводит выборы.

2. Круговой алгоритм:

• Каждый процесс знает следующего за ним в круговом списке.

• Когда процесс обнаруживает отсутствие координатора, он посылает следующему за ним процессу сообщение “ВЫБОРЫ” со своим номером.

• Если следующий процесс не отвечает, то сообщение посыпалось процессу, следующему за ним, и т.д., пока не найдется работающим процессом.

• Каждый работающий процесс добавляет в список работающих свой номер и переправляет сообщение дальше по кругу.

• Когда процесс обнаружит в списке свой собственный номер (круг пройден), он меняет тип сообщения на “КООРДИНАТОР” и оно проходит по кругу, извещая всех о списке работающих и координаторе.

Взаимное исключение

Речь о доступе к ресурсам. Рассмотрим несколько алгоритмов:

1. Централизованный алгоритм.

• Все процессы запрашивают у координатора разрешение на вход в критическую секцию и ждут этого разрешения.

• Координатор обслуживает запросы в порядке поступления.

• Получив разрешение процесс входит в критическую секцию.

• При выходе из нее он сообщает об этом координатору.

Кол-во сообщ. на одно прохождение критической секции — 3.

Недостатки алгоритма — крах координатора или его перегрузка сообщениями.

2. Алгоритм с круговым маркером.

• Каждый процесс знает следующего за ним в круговом списке.

• По колпаку циркулирует маркер, дающий право на вход в критическую секцию.

• Получив маркер (специальное сообщение) процесс либо входит в критическую секцию, либо переправляет маркер дальше.

• После выхода из критической секции маркер переправляется дальше.

3. Централизованный алгоритм на основе временных меток.

Необходимо упорядочение всех событий в сист. по времени.

Вход в критическую секцию:

• Когда процесс желает войти в критическую секцию, он посылает всем процессам сообщение-запрос, содержащее имя критической секции, номер процесса и текущее время.

• После посылки запроса процесс ждет, пока все дадут ему разрешение.

• После получения от всех разрешения, он входит в критическую секцию.

Поведение процесса при приеме запроса:

• Если получатель не находится внутри критической секции и не запрашивает разрешение на вход в нее, то он посыпает отправителю сообщение “OK”.

• Если получатель находится внутри критической секции, то он не отвечает, а запоминает запрос.

• Если получатель выдал запрос на входжение в эту секцию, но еще не вошел в нее, то он сравнивает временные метки своего запроса и чужого. Побеждает тот, чья метка меньше. Если чужой запрос победил, то процесс посыпает сообщение “OK”. Если у чужого запроса метка больше, то ответ не посыпается, а чужой запрос запоминается.

Выход из критической секции:

• После выхода из секции он посыпает сообщение «OK» всем процессам, запросы от которых он запомнил, а затем стирает все запомнившиеся запросы.

Количество сообщений на одно прохождение секции — $2(n - 1)$, где n — число процессов. Если какой-то процесс перестанет функционировать, то отсутствие разрешения от него всех остановит.

Если в централизованном алгоритме есть опасность перегрузки координатора, то в этом алгоритме перегрузка любого процесса приведет к тем же последствиям.

Координация процессов

1. Если известен и “потребитель” и “производитель”:

• сообщения точка-точка.

2. Если **НЕ**известен “потребитель”:

• широковещательные сообщения;

• сообщения в ответ на запрос.

3. Если **НЕ**известен и “потребитель” и “производитель” :

• сообщения и запросы через координатора;

• широковещательный запрос.

[Бахтин, Презы по распределенным системам, Конспект Лекций.zip/3-4-MPI-Sync.doc: page 8-15]

DOP 20 Понятие о парадигме программирования. Основные парадигмы программирования. Языки и парадигмы программирования.

<b

DOP 25 Промежуточные представления программы: абстрактное синтаксическое дерево; последовательность трехадресных инструкций. Базовые блоки и граф потока управления.

Промежуточное представление программы (Intermediate Representation, IR) — форма представления программы, ориентированная на удобство дальнейшей обработки компилятором. Различают следующие IR:

- HIR (высокий уровень) — абстрактное синтаксическое дерево(АСД) — конечное помеченное ориентированное дерево, в котором внутренние вершины сопоставлены (помечены) с операторами языка программирования, а листья — с соответствующими операндами. По сути результат парсинга программы на языке программирования, по АСД можно однозначно восстановить программу, по остальным уже нельзя.

- MIR (средний уровень) — включает в себя:

1. Инструкции

- присваивание: $x \leftarrow op y z$; $x \leftarrow op y$; $x \leftarrow y$; $x[i] \leftarrow y$; $x \leftarrow y[i]$;
- переходы: $goto L$; $ifTrue x goto L$; $ifFalse x goto L$;
- дополнительное: $param x$; $call x n$; $return y$;

2. таблицы символов,

- переменные, их имена в программе и атрибуты, такие как область видимости, тип, для имен функций - число параметров, и т. п.

- LIR (нижний уровень) — фактически машинные инструкции — используется для машинно-зависимых задач, например, распределение регистров и выбор подходящих команд.

Базовые блоки и граф потока управления

Базовым блоком (ББ или линейным участком) называется последовательность следующих одна за другой инструкций MIR, обладающая следующими свойствами:

- Поток управления может входить в базовый блок только через его первую инструкцию (т.е. в программе нет переходов в середину базового блока).

Грубо говоря, базовый блок содержит инструкции, которые будут выполнены (или не выполнены) обязательно все вместе, независимо ни от чего. Поэтому он базовый.

Чтобы выделить базовые блоки, достаточно найти все их начала (НББ):

- первая инструкция программы

- инструкция, на которой есть метка

- следующая инструкция после перехода

Граф потока управления (ГПУ) — граф, обладающий следующими свойствами:

- Вершины — базовые блоки.

- Дуга соединяет выход одного блока со входом другого, если второй блок может выполняться сразу следом за первым.

Если последняя инструкция базового блока — условный переход, то из этого блока будут выходить две дуги.

Если первая инструкция базового блока имеет метку, то в этот блок будут входить дуги из всех базовых блоков, у которых последняя инструкция — переход на эту метку.

Чтобы построить ГПУ, надо

1. выделить базовые блоки;

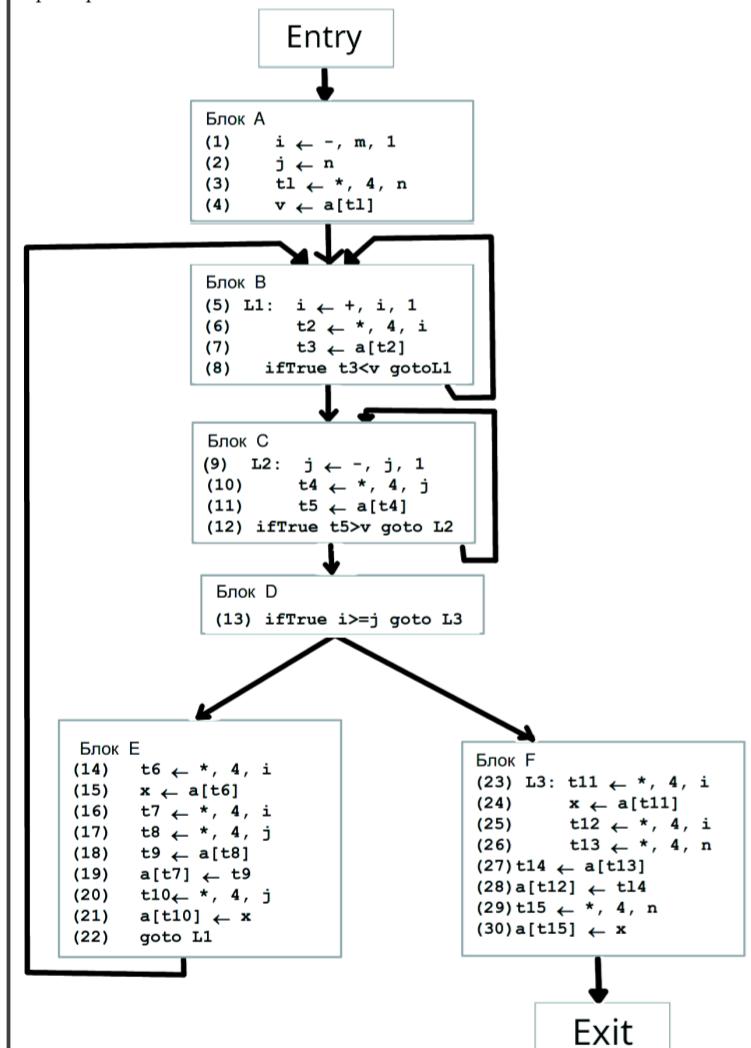
2. проанализировать дуги из блока туда, куда может пойти управление после этого блока. Определяется по последней инструкции:

- либо просто следующий блок

- либо это безусловный переход — туда где метка этого перехода

- либо и туда и туда, если это условный переход

Пример ГПУ:



[Презентации Гайсаряна, slide 7-28]

DOP 26 Промежуточные представления программы: глобальная оптимизация при компиляции программы. Построение передаточных функций базовых блоков. Монотонные и дистрибутивные передаточные функции. Метод неподвижной точки и его применение для нахождения достигающих определений.

Глобальная оптимизация — оптимизация в пределах процедуры (шире чем в базовом блоке).

К глобальным оптимизациям относятся, например:

- Удаление мертвого кода (вычисляющего неиспользуемые переменные).

- Устранение общих подвыражений (одинаковых по тексту выражений, которых не изменились значения переменных, а значит и результат)

- Вынос инвариантов цикла.

Для выполнения таких преобразований необходима информация, полученная с помощью анализа потоков данных. Он позволяет извлекать различные свойства, вычисленные вдоль путей программы, используя при этом общий алгоритм. Общие понятия для всех анализов:

- Точки программы $(\dots, p_j, p_{j+1}, p_{j+2}, \dots)$ расположены между её инструкциями $(\dots, I_j, I_{j+1}, \dots)$ и характеризуют соответствующие состояния программы.

• Состояние программы — множество значений всех переменных программы, включая переменные в кадрах стека времени выполнения, находящихся ниже текущей вершины стека.

- Инструкция программы I_j описывается парой состояний: состоянием в конце программы p_j перед инструкцией I_j (**входным состоянием**, $In[I_j]$) и состоянием в конце программы p_{j+1} после инструкции (**выходным состоянием**, $Out[I_j]$).

• Считается, что с каждой инструкцией I_j связаны две **передаточные функции**: передаточная функция прямого обхода ГПУ (от входного состояния) f_{I_j} и передаточная функция обратного обхода (от выходного до входного состояния) $f_{I_j}^b$. Передаточная функция определяет как изменяется состояние программы, когда встречается эта инструкция.

Т.е.: $Out[I_j] = f_{I_j}(In[I_j])$ при прямом обходе и $In[I_j] = f_{I_j}^b(Out[I_j])$ при обратном.

• Для ББ B из инструкций I_1, \dots, I_n по определению $In[B] = In[I_1], Out[B] = Out[I_n]$. Передаточная функция f_B блока B по определению равна композиции передаточных функций его инструкций: $f_B(x) = f_{I_n}(f_{I_{n-1}}(\dots f_{I_1}(x) \dots)) = (f_{I_1} \circ f_{I_2} \circ \dots \circ f_{I_n})(x)$, или $f_B = f_{I_1} \circ f_{I_2} \circ \dots \circ f_{I_n}$, и соответственно, $f_B^b = f_{I_n}^b \circ f_{I_{n-1}}^b \circ \dots \circ f_{I_1}^b$.

Итого, при прямом обходе: $Out[B] = f_B(Out[B])$; при обратном обходе: $In[B] = f_B^b(Out[B])$.

(Осторожнее с порядком функций в операции композиции \circ , есть разные мнения на этот счет. Здесь записано мнение Гайсаряна. В Википедии наоборот.)

Анализ достигающих определений Один из видов анализа потоков данных.

- Определением переменной x называется инструкция, которая присваивает значение переменной x .

- Использованием переменной x называется инструкция, одним из operandов которой является переменная x .

- Определение d **достигает** точки p , если существует путь от точки d , непосредственно следующий за d , к точке p , такой, что вдоль этого пути d остается живым.

- **Передаточные функции достигающих определений.** Рассмотрим инструкцию I

$$d : u = v + w$$

, расположенную между точками p_1 и p_2 программы. По определению передаточной функции $y = f_I(x)$ инструкция I сначала убивает все предыдущие определения u , а потом порождает d — новое определение u . Следовательно, $f_I(x) = gen_I \cup (x - kill_I)$, где x — состояние во входной точке инструкции I .

Пусть базовый блок B содержит n инструкций, каждая из которых имеет передаточную функцию $f_i(x) = gen_i \cup (x - kill_i)$, $i = 1, 2, \dots, n$. Тогда передаточная функция для базового блока B может быть записана как $f_B(x) = gen_B \cup (x - kill_B)$, где $kill_B = kill_1 \cup kill_2 \cup \dots \cup kill_n$ и $gen_B = gen_1 \cup (gen_{n-1} - kill_n) \cup \dots \cup (gen_1 - kill_2 - kill_3 - \dots - kill_n)$. Таким образом, для каждого базового блока B_i можно выписать уравнение: $Out[B_i] = f_B(Out[B_i])$.

В случае анализа достигающих определений: $Out[B_i] = gen_{B_i} \cup (In[B_i] - kill_{B_i})$.

$Pred(B)$ — множество всех вершин ГПУ, которые непосредственно предшествуют вершине B . Следовательно, $In[B_i] = \bigcup_{P \in Pred(B_i)} Out[P]$.

Произведя подстановку, получим систему уравнений:

$$In[B_i] = \bigcup_{P \in Pred(B_i)} (gen_P \cup (In[P] - kill_P)), \quad i = 1, 2, \dots, n$$

Монотонные и дистрибутивные передаточные функции

- Полурешетка это множество L , на котором определена бинарная операция «сбор» \wedge , такая, что $\forall x, y, z \in L$:

- $x \wedge x = x$ (идемпотентность)

- $x \wedge y = y \wedge x$ (коммутативность)

- $x \wedge (y \wedge z) = (x \wedge y) \wedge z$ (ассоциативность)

- Для всех пар $x, y \in L$ определим отношение $\leq: x \leq y$ тогда и только тогда, когда $x \wedge y = x$.

- Структурой потока данных называется четверка $\langle D, F, L, \wedge \rangle$, где D — направление анализа (Forward или Backward), F — семейство передаточных функций, L — поток данных (множество элементов полурешетки), \wedge — реализация операции сбора.

- Структура потока данных для **анализа достигающих определений**: $\langle Forward, GK, Def, \cup \rangle$, где GK — семейство передаточных функций вида $gen-kill$, Def — множество определений переменных.

- Структура потока данных $\langle D, F, L, \wedge \rangle$ называется **монотонной**, если $\forall x, y \in L, \forall f \in F (x \leq y) \Rightarrow f(x) \leq f(y)$.

- Структура потока данных $\langle D, F, L, \wedge \rangle$ называется **дистрибутивной**, если $\forall x, y \in L, \forall f \in F (f(x \wedge y) = f(x) \wedge f(y))$.

Метод неподвижной точки и его применение для поиска достигающих определений

(Обобщение, никто не называет это методом неподвижной точки, но видимо имелось виду это.)

Общий итеративный алгоритм решения задачи анализа потока данных:

Вход: граф потока управления, структура потока данных $\langle D, F, L, \wedge \rangle$, передаточная функция $f_B \in F$, константа $v \in L$ для граничного условия.

Выход: значения из L для $In[B]$ и $Out[B]$ для каждого блока B в графе потока.

- 1) $out[Byход] = \text{вход};$

- 2) $\text{for (каждый базовый блок } B, \text{ отличный от входного) } OUT[B] = \top;$

- 3) $\text{while (внесены изменения в } IN)$

- 4) $\text{for (каждый базовый блок } B, \text{ отличный от выходного) } \{$

- 5) $\text{OUT}[B] = \wedge_{S \text{ — предшественник } B} IN[S];$

- 6) $IN[B] = f_B(OUT[B]);$

a) Итеративный алгоритм для прямой задачи потока данных

- 1) $IN[\text{Выход}] = \text{выход};$

- 2) $\text{for (каждый базовый блок } B, \text{ отличный от выходного) } IN[B] = \top;$

- 3) $\text{while (внесены изменения в } IN)$

- 4) $\text{for (каждый базовый блок } B, \text{ отличный от выходного) } \{$

- 5) $OUT[B] = \wedge_{S \text{ — предшественник } B} OUT[S];$

- 6) $IN[B] = f_B(OUT[B]);$

b) Итеративный алгоритм для обратной задачи потока данных</

DOP 30 Потоки в сетях. Алгоритм построения максимального потока. Оценка сложности алгоритма.

• **Потоковая сеть** — набор $G = (V, E, c, s, t)$, где (V, E) — орграф без множественных дуг (в частности, $(u, v) \in E \implies (v, u) \notin E$), c — вектор пропускных способностей дуг, $|c| = |E|$, вершины s, t — исток и сток соответственно. Для каждой вершины u обозначим:

$$u_+ = \{v \in V \mid (u, v) \in E\}$$

$$u_- = \{v \in V \mid (v, u) \in E\}$$

• f — поток в G , т.е. вектор, $|f| = |E|$, для которого выполнены условия:

$$1. f(e) \in [0, c(e)], e \in E$$

$$2. \forall u \in V \setminus \{s, t\}, f_+(u) = \sum_{v \in u_+} f(u, v) = \sum_{v \in u_-} f(v, u) = f_-(u).$$

• $\|f\| = f_+(s)$ — величина потока в сети.

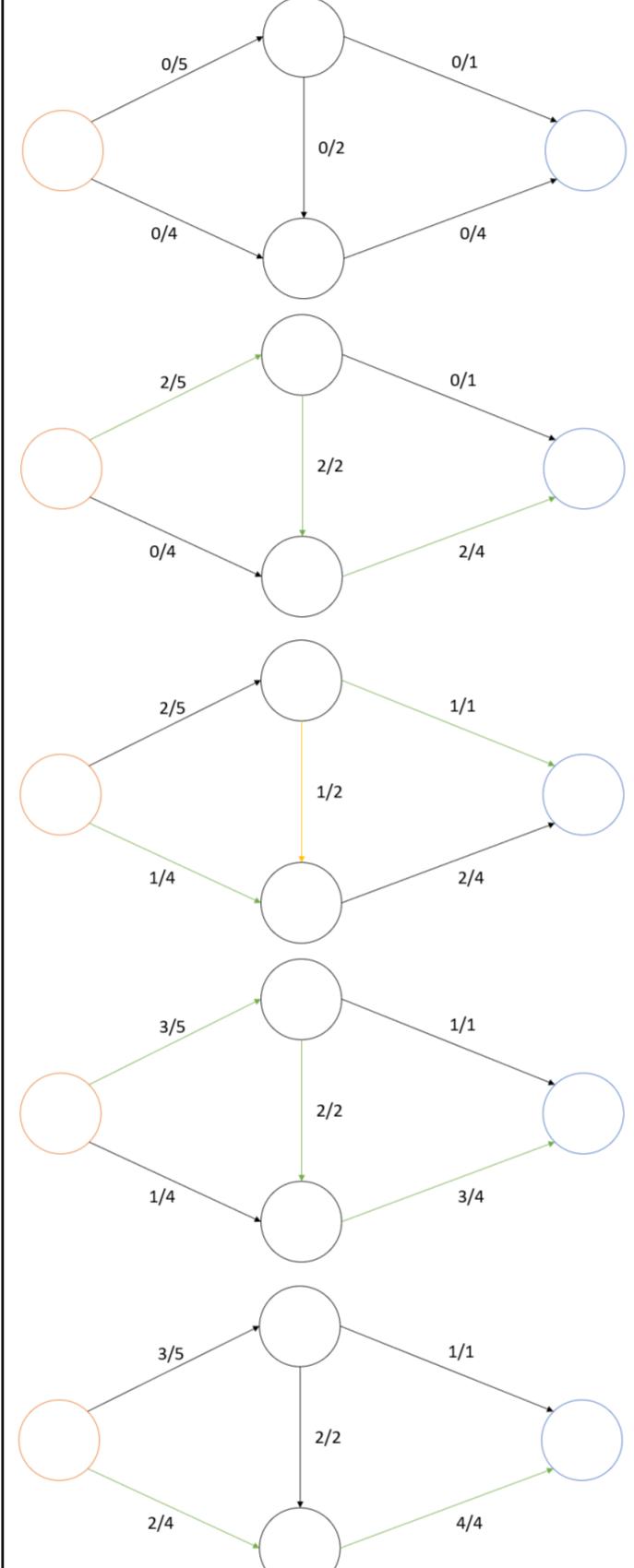
• **Задача поиска максимального потока:** Найти поток f максимальной величины при условиях 1., 2.

Алгоритм Форда-Фалкерсона

Инициализация. Для каждой дуги $(u, v) \in V$ добавим обратную дугу (v, u) , сделав на ней соответствующую пометку. Положим изначально $f(e) = 0 \forall e \in V$.

Шаг алгоритма. Пусть имеется некоторый вычисленный поток $f(e) \forall e \in V$. Определим "остаток пропускной способности" следующим образом: для прямых дуг $r(u, v) = c(u, v) - f(u, v)$, для обратных — $r(v, u) = f(u, v)$. Попытаемся найти какой-либо путь p из s в t , в котором у всех дуг остаток пропускной способности положителен. Если такого пути нет, то максимальный поток построен, и алгоритм завершается. Если он нашелся то вычислим остаток пропускной способности пути: $r(p) = \min\{r(e) \mid e \in p\}$; и обновим поток для каждой дуги в p : если дуга (u, v) прямая, то $f(u, v) = f(u, v) + r(p)$, если дуга (v, u) обратная, то $f(u, v) = f(u, v) - r(p)$.

Алгоритм гарантированно сходит при условии целочисленности пропускной способности с для каждой дуги. Для каждого найденного пути величина потока увеличивается хотя бы на 1, алгоритм поиска пути может быть любым, но, например, можно найти его при поиском в ширину со сложностью $O(|E|)$, и таким образом сложность алгоритма будет составлять $O(\|f\| |E|)$.



[Пупкин-Залупкин, *Напиши источник!*, page 69-96]

DOP 28 Постановка задачи дискретной оптимизации. Метод ветвей и границ. Задача целочисленного линейного программирования.

Постановка задачи дискретной оптимизации: найти $\max_{x \in X} f(x)$, где X — конечное или счетное (ми-во допуст. значений перем. x), в постановке м.б. и нахождение тах

Метод ветвей и границ

Две процедуры: ветвление и нахождение оценок (границ), т.е. для того чтобы МВГ работал, нужны:

1) метод разбиения задачи на подзадачи

2) функция оценки решения.

Процедура ветвления состоит в разбиении множества допустимых значений переменной x на подобласти (подмножества) меньших размеров. Процедуру можно рекурсивно применять к подобластям. Полученные подобласти образуют дерево, называемое **деревом поиска** или деревом ветвей и границ. Узлами этого дерева являются построенные подобласти (подмножества) множества значений переменной x . Процедура нахождения оценок заключается в поиске верхних и нижних границ для решения задачи на подобласти A дерева поиска больше, чем верхняя граница на какой-либо ранее просмотренной подобласти B , то A может быть исключена из дальнейшего рассмотрения (правило отсева). Обычно минимальную из полученных верхних оценок записывают в глобальную переменную m ; любой узел дерева поиска, нижня граница которого больше значения m , может быть исключен из дальнейшего рассмотрения (закрыт). Если нижняя граница для узла дерева совпадает с верхней границей, то это значение является минимумом функции на подобласти A дерева поиска больше, чем верхняя граница на каком-либо ранее просмотренной подобласти.

Основная задача линейного программирования (озЛП) (c_1, c_2, \dots, c_n), найти

$$\max_{x \in R, Ax \leq b} \langle c, x \rangle.$$

Каноническая задача ЛП:

$$\max_{Ax=b, x \geq 0} \langle c, x \rangle.$$

Формально данные задачи не являются дискретными, но они могут быть сведены к перебору конечного числа угловых точек (вершин полиддра, задающего ограничения) на основании принципа граничных решений:

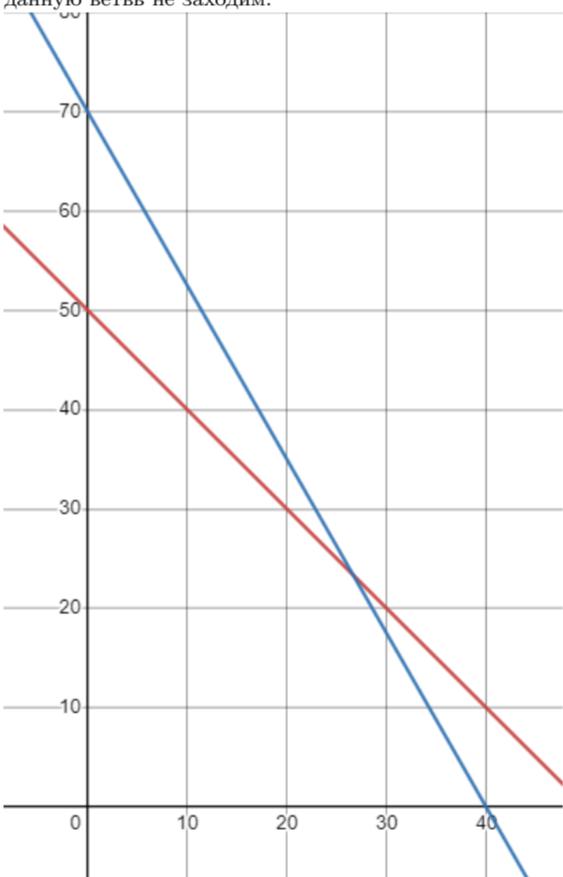
Если задача имеет решение, то найдется такая подматрица A_I матрицы A , что любое решение системы уравнений $A_I x = b_I$ реализует максимум.

Для задачи целочисленного ЛП на переменные накладывается требование их принадлежности мн-ву целых чисел:

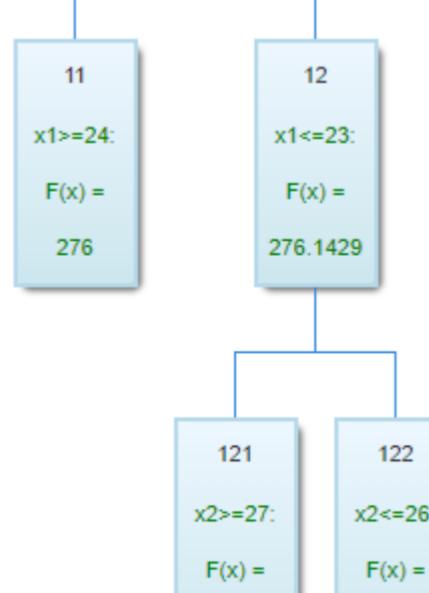
$$\max_{x \in Z, Ax \leq b} \langle c, x \rangle.$$

Пример: Найти $\max Z = 5x_1 + 6x_2$ при ограничениях $x_1 + x_2 \leq 50$, $4x_1 + 7x_2 \leq 280$, $x_1, x_2 \geq 0$ — целые.

Общая схема МВГ: для примера: строим область ограничения $x_1 \leq -x_2 + 50$ и $x_1 \leq -4x_2 + 70$, точка пересечения $x_1 = \frac{70}{3} = 23.33$, $x_2 = \frac{80}{3} = 26.66$, $Z = 5 * \frac{70}{3} + 6 * \frac{80}{3} = 276.667$ — не является целочислом. Решение лежит внутри области, заданной ограничениями. Мы проверяем другие конечные точки, проводя линии, задающие целочисленные ограничения для x_1, x_2 по области. Выбираем переменную для ветвления (ногой) выбора, тк в некоторых примерах придется рассмотреть больше/меньше ветвей, в данном случае x_2 становится параметром для МВГ: берем область $x_2 \leq 26$, находим точку пересеч. с границей $x_1 = 24$, $x_2 = 26$, $Z = 276 -$ получили целочисл. решение. Сама точка называется текущим целочисленным рекордом или просто рекордом, а оптимальное значение целочисленной задачи — текущим значением рекорда. Это значение является нижней границей оптимального значения исходной задачи и с ним будем сравнивать все след. значения Z , переходя к след. ветви $x_2 \geq 27 \Rightarrow \{x_1 = 22.75, x_2 = 27, Z = 275.75\}$ — не целочисл. и \leq рекорда, ветвимся теперь по x_1 : строим $x_1 \leq 23 \Rightarrow \{x_1 = 23, x_2 = 26.85, Z = 276.1429\} >$ рекорда, теперь внутри этой ветви нужно развернуться по $x_2 \leq 26$ и $x_2 \geq 27$, внутри этих ветвей получим значения целевой функции меньше или равные рекорду. Пробуем ветвь по $x_1 : x_1 \geq 24$ и приходим к $Z = 276 \leq$ рекорда, в данную ветвь не заходим.



Ограничения, сюда добавляем также ограничение текущей ветки



Ветвление по x_1

[Пупкин-Залупкин, *Напиши источник!*, page 69-96]

DOP 26 Локальная оптимизация при компиляции программы. Ориентированный ациклический граф и метод нумерации значений.

Базовый блок (ББ или линейным участком) называется последовательность следующих одна за другой инструкций МПР, обладающая следующими свойствами:

• Поток управления может входить в базовый блок только через его первую инструкцию (т.е. в программе нет переходов в середину базового блока),

• Поток управления покидает базовый блок без останова или ветвлений, кроме, возможно, последней инструкции базового блока.

Локальная оптимизация — это оптимизация, которая выполняется в пределах одного базового блока (ББ). Возможные локальные оптимизации:

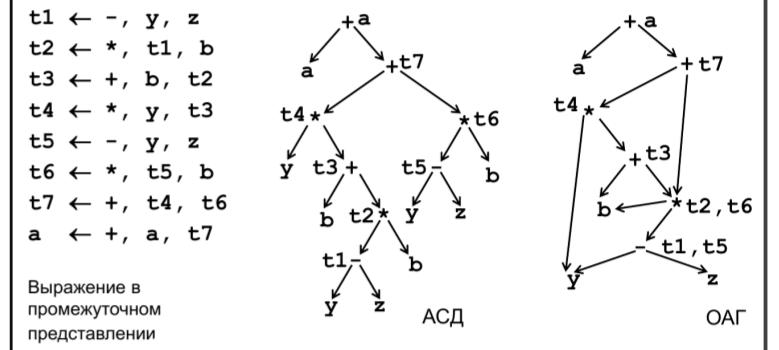
1. Удаление общих подвыражений (инструкций, повторно вычисляющих уже вычисленные значения).
2. Удаление мертвого кода (инструкций, вычисляющих значения, которые впоследствии не используются).
3. Сворачивание констант (вычисление константных выражений).
4. Изменение порядка инструкций, там, где это возможно, чтобы сократить время хранения временного значения на регистре.
5. Снижение стоимости вычислений (замена более дорогих операций более дешевыми).

ОАГ и метод нумерации значений

Все указанные преобразования для локальной оптимизации можно выполнить за один проход ББ, если представить его в виде ориентированного ациклического графа (ОАГ). Суть ОАГ заключается в том, что узлы, представляющие одинаковые значения, присутствуют в единственном экземпляре.

Пример. Выражение в исходном коде:

$$a = a + y * (b + (y - z) * b) + (y - z) * b$$



ОАГ можно представить в виде таблицы значений (вычислять таблицу пропись для понимания). Пример таблицы ниже.

Каждая строка таблицы значений представляет один узел ОАГ. Строки содержат:

1. свой номер (номер значения)
2. сигнатуру операции
- Для обычных операций `<op, #left, #right>`, где `op` — код операции, а `#left` и `#right` — номера значений левого и правого operandов (у unary операций `#right` равен 0)
- Унарные операции `id` и `nm` определяют соответственно имена переменных и константы (листовые узлы).
3. Имена переменных, в которых хранится это значение.

Алгоритм (на псевдокоде) построения ОАГ для базового блока B , содержащего n инструкций вида $t_i \leftarrow Op_i, l_i, r_i$.

Функция `#val(s)` определяет номер значения, определяемого сигнатурой $s = (Op, #val(1), #val(r))$.

```
for each "ti ← Opi, li, ri" do
    si = (Opi, #val(li), #val(ri))
    if T3 содержит sj == si
        then
            вернуть j в качестве значения #val(si)
        else
            завести в T3 новую строку T3k
            записать сигнатуру si в строку T3k
            вернуть k в качестве значения #val(si)
    t1 ← -, y, z          t15 ← -, y3, z4          t1 ← -, y, z
    t2 ← *, t1, b          t26 ← *, t15, b2          t2 ← *, t1, b
    t3 ← +, b, t2          t37 ← +, b3, t26          t3 ← +, b, t2
    t4 ← *, y, t3          t48 ← *, y2, t37          t4 ← *, y, t2
    t5 ← -, y, z          t59 ← -, y3, z4          t5 ← t1
    t6 ← *, t5, b          t610 ← *, t59, b2          t6 ← t2
    t7 ← +, t4, t6          t711 ← +, t48, t610          t7 ← +, t4, t6
    a ← +, a, t7          a12 ← +, a1, t79          a ← +, a, t7
```

(a) Блок Е до оптимизации

(b) Блок Е после нумерации значений

(c) Блок Е после оптимизации

1	id	ссылка в ТС	a
2	id	ссылка в ТС	b
3	id	ссылка в ТС	y
4	id	ссылка в ТС	z
5	-	3	t1, t5
6	*	5	t2, t6
7	+	2	t3
8	*	3	t4
9	+	8	t7
10	+	1	a
# значения	КОП	# операнда	# операнда
			Присоединенные переменные
			Определение значения (сигнтура)

При нумерации значений (и восстановлении базового блока из ОАГ) автоматически получаем удаление общих подвыражений. Для значений, которым соответствуют несколько переменных достаточно вычислить одну из них, а во вторую скопировать результат.

Сворачивание констант также можно провести во время нумерации значений. Если оба операнда — константы, их результат можно сразу вычислить и записать в таблицу значений как константу.

Удаление мертвого кода — более сложная оптимизация, которая требует знаний о других блоках. Для описания алгоритма расширим понятие базового блока

Базовым блоком (ББ) называется тройка $(B = P, Input, Output)$, где

- P — последовательность инструкций,
- $Input$ — множество переменных, определенных до входа в блок B ,
- $Output$ — множество переменных, используемых после выхода из блока B .

Живыми называются переменные, значения которых, вычисленные в рассматриваемом базовом блоке, используются в других базовых блоках.

Пример. Рассмотрим базовый блок $B = \langle P, \{a, b, c, d\}, \{a, b\} \rangle$



Список литературы