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 комплексногопеременного. 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, равенство 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 и распределенной памятью. Производительность вычислительных систем, методы оценки и измерения.
osn 23. Основные методы обработки изображений: тональная коррекция, свёртка изображений, выделение краёв.
osn 24. Линейные обыкновенные дифференциальные уравнения и системы. Фундаментальная система решений. Определитель Вронского.
osn 25. Теоремы существования и единственности решения задачи Коши для обыкновенного дифференциального уравнения первого порядка, разрешенного относительно производной.
osn 26. Функции алгебры логики. Реализация их формулами. Совершенная дизъюнктивная нормальная форма.
osn 27. Схемы из функциональных элементов и простейшие алгоритмы их синтеза. Оценка сложности схем, получаемых по методу Шеннона.
osn 28. Вероятностное пространство. Случайные величины. Закон больших чисел в форме Чебышева.
osn 29. Квадратурные формулы прямоугольников, трапеций и парабол.
osn 30. Методы Ньютона и секущих для решения нелинейных уравнений.
osn 31. Численное решение задачи Коши для обыкновенных дифференциальных уравнений. Примеры методов Рунге-Кутта.
osn 32. Задача Коши для уравнения колебания струны. Формула Даламбера.
osn 33. Постановка краевых задач для уравнения теплопроводности. Метод разделения переменных для решения первой краевой задачи.

Я равномерно схожусь к нежеланию ботать.

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
Он, пытаясь ее впечатлить: Официант, принесите нам два килограмма говна, пожалуйста

– Синус, косинус, тангенс, котангенс, 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 астрологи обнаружили, что в России уже 22 года продолжается год Крысы.

Маленький одноногий 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.
Долбаёб – ловко парировали 128 осьминогов.

Монеточка парню перед сексом:
– Выбирай, орел или решка?

Однажды в студеную зимнюю пору лошадка пиписькой примерзла к забору.

В дверь постучались 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 переименуется в Организацию Обеспокоенных Наций.

– А вот когда умирает черепашка, проносится ли у нее жизнь перед глазами или типа 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, в дверь постучали, в дверь постучали, в дверь постучали, в дверь постучали, в дверь постучали, в дверь постучали, 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 останавливало только то, что в городе не было порта.

А спонсор этого дня — батюшка на батуте, вывпивший перед этим два литра пива.
Батюшка на батуте, выпивший перед эти два литра пива — поп-рыгун

В дверь постучали 256 раз
– 32 осьминога – подумал Штрилиц
– Заебал впусти – кричал Мюллер

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

Okko откроет российский аналог Pornhub “Чпокко”.

В дверь вежливо постучали ногой.
– Безруков! – догадался Штирлиц.

Минкульт: в честь 9 мая будет издан ремейк знаменитой военной поэмы: “Насилий Мародеркин”.

Спрашивают у бывшей проститутки: "Как вам удалось стать миллионершей?"
– 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.

Анекдот от Никитина:
Грин работал у отца на ферме, а когда отец умер – занялся математикой и спился. Можете рассказать это в 6 билете.

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

Россия объявила о победе в конкурсе “Евроненавидение”.

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

Делу 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 знаю”

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.

По разные стороны Москвы – XXI век

По разные стороны Москвы – XXI век

dop 1. Теорема Поста о полноте систем функций в алгебре логики.
dop 2. Графы, деревья, планарные графы; их свойства. Оценка числа деревьев.
dop 3. Логика 1-го порядка. Выполнимость и общезначимость. Общая схема метода резолюций.
dop 4. Логическое программирование. Декларативная семантика и операционная семантика; соотношение между ними. Стандартная стратегия выполнения логических программ.
dop 5. Сортировка. Простейшие алгоритмы – сортировка выбором, вставками, обменом. Оценка сложности алгоритмов сортировки. Быстрая сортировка и ее сложность в среднем и в наихудшем случаях.
dop 6. Язык ассемблера как машиннозависимый язык низкого уровня. Организация ассемблерной программы, секции кода и данных (на примере ассемблера nasm или masm). Основные этапы подготовки к счёту ассемблерной программы: трансляция, редактирование внешних связей (компоновка), загрузка.
dop 7. Операционные системы. Управление оперативной памятью в вычислительной системе. Алгоритмы и методы организации и управления страничной оперативной памятью.
dop 8. Зависимости в реляционных отношениях: функциональные, многозначные, проекции/соединения. Проектирование реляционных БД на основе принципов нормализации отношений. Нормальные формы.
dop 9. Закон Амдала, его следствия. Граф алгоритма. Критический путь графа алгоритма, ярусно-параллельная форма графа алгоритма. Этапы решения задач на параллельных вычислительных системах.
dop 10. Глобальные и локальные модели освещения в компьютерной графике. Модель Фонга.
dop 11. Классификация языков, определяемых конечными автоматами, регулярными выражениями и праволинейными грамматиками. Эквивалентность и минимизация конечных автоматов.
dop 12. Функции FIRST и FOLLOW. LL(l)-грамматики. Конструирование таблицы предсказывающего анализатора.
dop 13. Жизненный цикл программного обеспечения (ПО). Основные виды деятельности при разработке ПО. Каскадная и итерационная модели жизненного цикла.
dop 14. Качество программного обеспечения и методы его контроля. Тестирование и другие методы верификации.
dop 15. Основные понятия криптографии. Односторонняя функция с секретом. Протокол Диффи-Хеллмана выработкиобщего секретного ключа по открытому каналу связи.
dop 16. Основные принципы построения и архитектура сети Интернет. Алгоритмы и протоколы внешней и внутренней маршрутизации. Явление перегрузки и методы борьбы с ней.
dop 17. Теоретические основы передачи данных, физический уровень стека протоколов. Системы передачи данных Ethernet и Wi-Fi: алгоритмы работы, управление множественным доступом к каналу
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.
dop 19. Понятие о парадигме программирования. Основные парадигмы программирования. Языки и парадигмы программирования.
dop 20. Основные характеристики функциональных языков программирования. Использование понятий функционального программирования (замыкания, анонимные функции) в современных объектно-ориентированных языках.
dop 21. Синхронизация в распределенных системах. Синхронизация времени. Логические часы. Выборы координатора. Взаимное исключение. Координация процессов.
dop 22. Отказоустойчивость в распределенных системах. Типы отказов. Фиксация контрольных точек и восстановление после отказа. Репликация и протоколы голосования. Надежная групповая рассылка.
dop 23. Распределенные файловые системы. Доступ к директориям и файлам. Семантика одновременного доступа к одному файлу нескольких процессов. Кэширование и размножение файлов.
dop 24. Промежуточные представления программы: абстрактное синтаксическое дерево; последовательность трехадресных инструкций. Базовые блоки и граф потока управления.
dop 25. Локальная оптимизация при компиляции программы. Ориентированный ациклический граф и метод нумерации значений.
dop 26. 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.
dop 27. Постановка задачи дискретной оптимизации. Метод ветвей и границ. Задача целочисленного линейного программирования.
dop 28. Комбинаторные методы нахождения оптимального пути в графе.
dop 29. 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 $(x_1, \dots, x_m)$ m чисел $x_1, \dots, x_m$ наз-ся **m-мерным координатным пространством** A_m.

$\rceil$ имеется некоторое множество M и некоторая функция $\rho : M \times M \to R^+$. 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. $\rho(x, y) \le \rho(x, z) + \rho(z, y)$ (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 **предельным значением функции** $u = f(M)$ в точке $A \in R^m$ (пределом функции при $M \to A$), если для $\forall$ сходящейся к A последовательности $M_1, \dots, M_n, \dots$ точек множества $\{M\}$, где $M_n \neq A$, соответствующая последовательность $f(M_1), \dots, f(M_n), \dots$ значений функций сходится к b.

Предел по Коши. Число $b \in R$ называется **предельным значением функции** $u = f(M)$ в точке $A = (a_1, \dots, a_m)$, если $\forall \varepsilon > 0\ \exists \delta :\ \forall M \in \{M\}$, удовлетворяющих $0 < \rho(M, A) < \delta$, выполняется $|f(M) - b| < \varepsilon$.

Теорема об эквивалентности определений предела. Определения предела функции по Коши и по Гейне эквивалентны.
▲ (Г $\implies$ К) $\rceil$ b – предел $u = f(M)$ в т. A по Гейне, но опр. по Коши не выполнено $\implies \exists \varepsilon > 0 : \forall \delta > 0\ \exists M \in \{M\} : 0 < \rho(M, A) < \delta, |f(M) - b| \ge \varepsilon \implies$ для $\delta_n = \frac{1}{n}$ $\exists M_n : 0 < \rho(M_n, A) < \delta_n,\ |f(M_n) - b| \ge \varepsilon \implies \{M_n\} \to A \implies$ по Гейне $\{f(M_n)\} \to b \implies$ противоречие с $|f(M_n) - b| \ge \varepsilon$.
(К $\implies$ Г) $\rceil$ b – предел $u = f(M)$ в т. A по Коши и $\{M_n\} \to A$. Фиксируем $\varepsilon > 0$, по Коши $\exists \delta > 0 : \forall M \in \{M\} : 0 < \rho(M, A) < \delta,\ |f(M) - b| < \varepsilon$. Т.к. $\{M_n\} \to A$, то для этого δ $\exists N \in N : \forall n \ge N,\ 0 < \rho(M_n, A) < \delta \implies |f(M_n) - b| < \varepsilon \implies \{f(M_n)\} \to b$ ■

Последовательность $M_1, \dots, M_n$ наз-ся **фундаментальной**, если $\forall \varepsilon > 0\ \exists N = N(\varepsilon) \in \mathbb{N} : \forall m \ge N, p \in \mathbb{N}$ выполнено $\rho(M_{m+p}, M_m) < \varepsilon$.
Критерий Коши сходимости посл-ти: последовательность $M_1, \dots, M_n$ сходится $\Longleftrightarrow$ последовательность фундаментальна.

Функция $f(M)$ **удовлетворяет в точке M условию Коши**, если $\forall \varepsilon > 0\ \exists \delta : \forall M', M'' \in \mathring{U}(M)$, 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.
▲ ($\implies$) $\rceil$ $\lim\limits_{M \to A} f(M) = b$. Выберем $\varepsilon > 0 \implies$ по опр. предела по Коши для $\frac{\varepsilon}{2}$ $\exists \delta > 0,\ \forall M', M'' \in \{M\} : 0 < \rho(M', A) < \delta,\ 0 < \rho(M'', A) < \delta \implies \left|f(M') - b\right| < \frac{\varepsilon}{2},\ \left|f(M'') - b\right| < \frac{\varepsilon}{2}$. Тогда $\left|f(M') - f(M'')\right| = \left|(f(M') - b) - (f(M'') - b)\right| \le \left|f(M') - b\right| + \left|f(M') - b\right| < \frac{\varepsilon}{2} + \frac{\varepsilon}{2} < \varepsilon$
($\Longleftarrow$) $\rceil$ $f(M)$ удовл. в т. A усл. Коши, $\{M_n\} : \{M_n\} \to A,\ M_n \neq A$. Выберем $\varepsilon > 0$ и соотв. $\delta > 0$ такое, что вып. усл. Коши, для этого δ. $\exists N \in \mathbb{N} : \forall n \ge N \implies 0 < \rho(M_n, A) < \delta$ (т.к. $\{M_n\} \to A$). Таким образом для $p = 1, 2, \dots \implies 0 < \rho(M_{n+p}, A) < \delta$ при $n \ge N \implies$ в силу усл. Коши $|f(M_{n+p}) - f(M_n)| < \varepsilon \implies \{f(M_n)\}$ – фундаментальна $\implies \{f(M_n)\}$ сход. к некоторому b.
$\rceil$ $\{M_n\} \to A,\ \{M'_n\} \to A,\ \{f(M_n)\} \to b,\ \{f(M'_n)\} \to b'$. Тогда $f(M_1), f(M'_1), \dots, f(M_n), f(M'_n), \dots$ сходится $\implies$ все её подпосл-ти сходятся к одному пределу $\implies b = b'$. ■

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

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

$\rceil$ функции $x_1 = \phi_1(t_1, \dots, t_k),\ \dots,\ x_m = \phi_m(t_1, \dots, t_k)$ заданы на множестве $\{N\}$ евклидова пространства E_m, $t_1, \dots, t_k$ — координаты точек в $E_k \implies \forall N(t_1, \dots, t_k) \in \{N\}$ ставится в соответствие точка $M(x_1, \dots, x_m)$ евклидова пространства E_m. $\rceil$ $\{M\}$ — множество всех этих точек, $u = f(x_1, \dots, x_m)$ - функция m переменных, заданная на $\{M\} \implies$ на множестве $\{N\}$ пространства E_k **определена сложная функция** $u = f(\phi_1(t_1, \dots, t_k), \dots, \phi_m(t_1, \dots, t_k)) = f(x(t))$

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

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

Теорема о сохранении знака. $\rceil$ $f(x)$ определена на мн-ве $\{X\}$, непрерывна в т. a и $f(a) > 0$ ($f(a) < 0$). Тогда $\exists \delta > 0 : \forall x \in \{X\}, x \in (a - \delta, a + \delta) \implies f(x) > 0$ ($f(x) < 0$).
▲ $\forall \varepsilon > 0\ \exists \delta(\varepsilon) > 0 : \forall x \in X,\ 0 < |x - a| < \delta \implies |f(x) - f(a)| < \varepsilon$. $\rceil$ $\varepsilon = \frac{|f(a)|}{2} \implies -\varepsilon < f(x) - f(a) < \varepsilon \implies f(a) - \frac{|f(a)|}{2} < f(x) < f(a) + \frac{|f(a)|}{2}$ (тот же знак) ■

Теорема о прохождении через 0. $\rceil$ $f(x)$ непрерывна на $[a, b]$, $f(a) > 0; f(b) < 0$. Тогда $\exists c \in (a, b) : f(c) = 0$.
▲ $\rceil$ $f(a) < 0,\ f(b) > 0,\ A = \{x \in [a, b] : f(x) < 0\}$. $A \neq \varnothing$ (т.к. $a \in A$) и ограничено сверху (например, числом b) $\implies \exists supA = c$. Покажем, что $f(c) = 0$.
$\rceil$ $f(c) > 0$. Тогда $c \neq a$ и по т. о сохр. знака $\exists \delta > 0 : f(x) > 0\ \forall x \in (c - \delta, c] \implies c \neq supA \implies$ противоречие $\implies f(c) \le 0$.
$\rceil$ $f(c) < 0$. Тогда $c \neq b$ и по т. о сохр. знака $\exists \delta > 0 : f(x) < 0\ \forall x \in [c, c + \delta) \implies c \neq supA \implies$ противоречие $\implies f(c) = 0$. ■

Теорема о достижении значения. $\rceil$ $f(x)$ непрерывна на $[a, b]$, тогда $\forall \gamma \in [\alpha, \beta]$, 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. $\rceil$ $\alpha < \gamma < \beta$. 👁 $g(x) = f(x) - \gamma$. Она удовл. усл. пред. теоремы $\implies \exists c \in [a, b] : g(c) = 0$, т.е. $f(c) = \gamma$ ■

Теорема Больцано-Вейерштрасса (нужна ниже)
Из любой ограниченной последовательности $\{x_n\}$ можно выделить сходящуюся подпоследовательность.
▲ $\rceil$ $\{X\}$ – мн-во значений последовательности $\{x_n\}$. Если $\{X\}$ – конечно, то найдется подпосл-ть такая, что $x_{n_1} = x_{n_2} = x_{n_3}$. Если $\{X\}$ – бесконечно, то по принципу Больцано-Вейерштрасса (у любого огр. беск. мн-ва есть хотя бы 1 предельная точка) у $\{X\}$ есть предельная точка $\implies \exists$ 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 $\{x_{k_n}\}$, предел c которой $\in [a, b]$. Очевидно, что посл-ть $\{f(x_{k_n})\}$ беск. большая, но в силу непр-ти функции в т. c эта посл-ть должна сходится к $f(c) \implies$ 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 $[a, b] \implies$ она огр. на $[a, b] \implies \exists M, m$ – ТВГ и ТНГ $f(x)$ на $[a, b]$. $\rceil$ $f(x) < M\ \forall x \in [a, b]$. Введем $g(x) = \frac{1}{M - f(x)}$. $g(x)$ – непр. на $[a, b]$, причем знаменатель не обр. в 0 $\implies$ огр. на $[a, b] \implies \exists A > 0 : \frac{1}{M - f(x)} \le A\ \forall x \in [a, b] \implies M - f(x) \ge \frac{1}{A} \implies f(x) \le M - \frac{1}{A}\ \forall x \in [a, b] \implies M \neq supf(x) \implies$ противоречие (для ТНГ аналогично) ■

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

Теорема о равномерной непрерывности (Кантора). Непрерывная на сегменте $[a, b]$ функция равномерно непрерывна на нем.
▲ $\rceil$ $f(x)$ непр. на $[a, b]$, но не р/н на нем. Тогда $\exists x'_n, x''_n \in [a, b] : \left|x'_n - x''_n\right| < \frac{1}{n} \forall n \in N$, но $\left|f(x'_n) - f(x''_n)\right| \ge \varepsilon$.
$\{x_n\}$ – огр. $\implies \exists \{x'_{n_k}\} \in [a, b] : \exists \lim\limits_{k \to \infty} x'_{n_k} = c$. 👁 $\{x''_{n_k}\}$: $\left|x''_{n_k} - c\right| \le \left|x''_{n_k} - x'_{n_k}\right| + \left|x'_{n_k} - c\right| \implies \{x''_{n_k}\} \to c$. По определению по Гейне непрерывности в точке: $\{f(x'_{n_k})\} \to f(c), \{f(x''_{n_k})\} \to f(c)$ – противоречие с $\left|f(x'_n) - f(x''_n)\right| \ge \varepsilon$. ■

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

Определения:
$\rceil$ $f(x)$ задана на $[a, b]$, $a < b$, T — разбиение $[a, b]$: $a = x_0 < x_1 < \dots < x_n = b$ на n частичных сегментов $[x_0, x_1],\ \dots,\ [x_{n-1}, x_n]$. $\rceil$ ξ_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 $I\{x_i, \xi_i\} = f(\xi_1)\Delta x_1 + f(\xi_2)\Delta x_2 + \dots + f(\xi_n)\Delta x_n = \sum\limits_{i=1}^{n} f(\xi_i)\Delta x_i$ 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 $\delta \to 0$, 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 $|I\{x_i, \xi_i\} - I| < \varepsilon$.

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

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

$\rceil$ $f(x)$ 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 $S = \sum\limits_{i=1}^{n} M_i \Delta x_i$ и $s = \sum\limits_{i=1}^{n} m_i \Delta x_i$ называются **верхней и нижней суммами** $f(x)$ для данного T сегмента $[a, b]$.

$\rceil$ $\overline{I}$ — точная нижняя граница множества $\{S\}$ верхних сумм, $\underline{I}$ — точная верхняя граница множества $\{s\}$ нижних сумм: $\overline{I} = inf\{S\}$, $\underline{I} = sup\{s\}$. Числа $\overline{I}$ и $\underline{I}$ — **верхний и нижний интегралы Дарбу** от $f(x)$.

Теоремы:
Необходимое условие интегрируемости – ограниченность. Неограниченная на $[a, b]$ функция $f(x)$ не интегрируема на $[a, b]$.
▲ Функция f(x) не ограничена на каком-либо промежутке $[x_{k-1}, x_k]$ $\implies$ $\forall$ разбиения слагаемое $f(\xi_k)\Delta x_k$ можно сделать сколь угодно большим $\implies \nexists lim$ ■

Лемма Дарбу. Нижний и верхний интегралы Дарбу $\overline{I}$ и $\underline{I}$ от $f(x)$ по $[a, b]$ являются соответственно пределами верхних и нижних сумм при $\Delta \to 0$.
▲ При $f(x) = const$ – очевидно. $\rceil$ $f(x) \neq const$, $M = \sup\limits_{[a,b]} f(x) > \inf\limits_{[a,b]} f(x) = m$. Фикс. $\varepsilon > 0$, $\exists$ разбиение $T^* = x^*_k$, $0 < k < l$ – разбиение на $[a, b]$, такое что $S(T^*) - \overline{I} < \frac{\varepsilon}{2}$
Положим $\delta = \frac{\varepsilon}{2(M - m)(l - 1)} > 0$ (δ зависит только от ε). $\rceil$ T – произвольное разбиение $[a, b]$. $T' = T \cup T^*$, тогда $0 \le S(T) - S(T') \le (M - m)\Delta_T(l - 1) < (M - m)(l - 1)\delta = \frac{\varepsilon}{2}$, Δ_T - диаметр разбиения $= \max\limits_{1 \le k \le n} \Delta x_k$, $\Delta_T < \delta$. Получаем, что $\forall \varepsilon > 0\ \exists \delta = \delta(\varepsilon) > 0$ такая что $\forall T$ - разбиения $[a, b] \implies 0 \le S(T) - \overline{I} = (S(T) - S(T')) + (S(T') - \overline{I}) \le \frac{\varepsilon}{2} + (S(T^*) - \overline{I}) \le \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon$ ■

Критерий Римана интегрируемости функции. $\rceil$ $f(x)$ определена и ограничена на $[a, b]$. $f \in \mathcal{R}[a, b] \iff \forall \varepsilon > 0\ \exists T$ – разбиение $[a, b]$, такое что $S(T) - s(T) < \varepsilon$.
▲
($\Rightarrow$) Пусть $f(x)$ интегрируема на $[a, b]$. Тогда по определению интеграла $\forall \varepsilon > 0\ \exists \delta = \delta(\varepsilon)$: для $\forall$ размеченного разбиения V сегмента $[a, b]$, для которого $\Delta_V < \delta$, выполнено: $|I - \sigma(V)| < \frac{\varepsilon}{3}$. Или, что то же самое:$I - \frac{\varepsilon}{3} < \sigma(V) < I + \frac{\varepsilon}{3}$. Тогда для верхняя и нижняя суммы Дарбу тоже лежат в этом промежутке (так как являются точными нижней и верхней гранями). Отсюда: $|S(T) - s(T)| \le \frac{2\varepsilon}{3} < \varepsilon$.
($\Leftarrow$) Пусть $\forall \varepsilon > 0 \exists T$ - разбиение сегмента $[a, b]$, такое что $|S(T) - s(T)| < \varepsilon$. Так как $s(T) \le \underline{I} \le \overline{I} \le S(T)$, то $\overline{I} - \underline{I} < \varepsilon$. ε - произвольное, $\Rightarrow I = \overline{I} = \underline{I}$. Для $\forall$ размеченного разбиения V сегмента $[a, b], \Delta_V < \delta$, выполнено: $S(T(V)) - s(T(V)) < \varepsilon$.
Так как $s(T(V)) \le \sigma(V) \le S(T(V))$ и $s(T(V)) \le I \le S(T(V))$, то $|I - \sigma(V)| < \varepsilon$ для любого размеченного разбиения V сегмента $[a, b]$. Мы доказали, что $I = \lim\limits_{\Delta_V \to 0} \sigma(V)$. Это означает, что функция $f(x)$ интегрируема на сегменте $[a, b]$ и $I = \int\limits_a^b f(x)dx$
■

Свойства определённого интеграла:

1. $\int\limits_a^a f(x)dx = 0$
2. $\int\limits_a^b f(x)dx = -\int\limits_b^a f(x)dx$
3. $\rceil$ $f(x)$ и $g(x)$ интегрируемы на $[a, b]$. Тогда $f(x) + g(x)$, $f(x) - g(x)$ и $f(x) \cdot g(x)$ также интегрируемы на $[a, b]$, причём $\int\limits_a^b [f(x) \pm g(x)]dx = \int\limits_a^b f(x)dx \pm \int\limits_a^b g(x)dx$
4. Если $f(x)$ интегрируема на $[a, b]$, то $cf(x)$ ($c = const$) тоже: $\int\limits_a^b cf(x)dx = c\int\limits_a^b f(x)dx$
5. Если $f(x)$ интегрируема на $[a, b]$, то $|f(x)|$ тоже.
6. $\rceil$ $f(x)$ интегрируема на $[a, b]$. Тогда $f(x)$ интегрируема на $\forall [c, d] \subset [a, b]$
7. $\rceil$ $f(x)$ интегрируема на сегментах $[a, c]$ и $[c, b]$. Тогда $f(x)$ интегрируема на $[a, b]$, причём $\int\limits_a^b f(x)dx = \int\limits_a^c f(x)dx + \int\limits_c^b f(x)dx$

Основная формула интегрального исчисления.
Первообразной функции $f(x)$ называется дифференцируемая функция $F(x)$: $F'(x) = f(x)$ на всей области определения $f(x)$.
Теорема. $\rceil$ $f \in \mathcal{R}[a, b]$, $F \in \mathcal{C}[a, b]$, $\forall x \in [a, b]$ $F'(x) = f(x)$. Тогда $\int_a^b f(x)dx = F(b) - F(a) = F(X)\Big|_a^b$
▲ x_k – $\forall$ разбиение, $F(b) - F(a) = (F(b) - F(x_{k-1})) + (F(x_{k-1}) - F(x_{k-2})) + \dots + (F(x_1) - F(a)) = \dots$ {F удовлетворяет всем условиям теоремы Лагранжа (непрерывна на [ ] и дифф-ма на ( ))} $\dots = f(\xi_1)(b - x_{k-1}) + f(\xi_2)(x_{k-1} - x_{k-2}) + \dots + f(\xi_k)(x_1 - a) \to \int_a^b f(x)dx$ при $\Delta \to 0$ ■

[И.В. Садовничая, *Определенный интеграл*, page 14-23]

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

Определения:

- $\rceil$ на числовой прямой E_1 или в m-мерном евклидовом пространстве E_m задано некоторое множество $\{x\}$. Если каждому натуральному числу n ставится в соответствие по определённому закону некоторая функция $f_n(x)$, определённая на множестве $\{x\}$, то множество занумерованных функций $f_1(x), f_2(x), \dots, f_n(x), \dots$ называется **функциональной последовательностью.**

- 👁 функциональную последовательность $\{u_n(x)\}$, с областью определения $\{x\}$. Формально написанная сумма

$$\sum_{n=1}^{\infty} u_n(x) = u_1(x) + u_2(x) + \dots + u_n(x) + \dots$$

бесконечного числа членов указанной функциональной последовательности называется **функциональным рядом.** Функции $u_n(x)$ называются **членами рассматриваемого ряда**, а множество $\{x\}$, на котором определены эти функции, называется **областью определения** этого ряда.

- $\rceil$ функциональная последовательность $f_1(x), f_2(x), \dots, f_n(x), \dots$ сходится на множестве $\{x\}$ пространства E_m к предельной функции $f(x)$, т.е. сходится в каждой его точке. Последовательность **сходится к функции** $f(x)$ **равномерно** (обозн. $\rightrightarrows$) на множестве $\{x\}$, если $\forall \varepsilon > 0$ $\exists$ номер $N(\varepsilon)$ такой, что $\forall n$, удовлетворяющих $n \geqslant N(\varepsilon)$, и $\forall x \in \{x\}$ справедливо неравенство $|f_n(x) - f(x)| < \varepsilon$.

- Функциональный ряд называется **равномерно сходящимся** на множестве $\{x\}$ к сумме $S(x)$, если последовательность $\{S_n(x)\}$ его частичных сумм сходится равномерно на множестве $\{x\}$ к предельной функции $S(x)$.

Теоремы:

Критерий Коши для функциональных последовательностей. $\{f_n(x)\} \rightrightarrows$ на $\{x\} \iff \forall \varepsilon > 0\ \exists N : \forall n \ge N\ \forall p \in \mathbb{N} : |f_{n+p}(x) - f_n(x)| < \varepsilon$ выполнено $\forall x \in \{x\}$
▲ $\Rightarrow$: $f(x) \equiv \lim\limits_{n \to \infty} f_n(x)$, $\forall \varepsilon > 0\ \exists N : \forall n \ge N\ \forall x \in \{x\}$:
$|f_n(x) - f(x)| < \varepsilon$ и $|f_{n+p}(x) - f(x)| < \varepsilon\ \forall p \in \mathcal{N} \implies |f_{n+p}(x) - f_n(x)| \le |f_{n+p}(x) - f(x)| + |f(x) - f_n(x)| < 2\varepsilon$
$\Leftarrow$: Заметим, что данное условие для $\forall$ фиксированного $x \in \{x\}$ означает сх. $\{f_n(x)\}$ в этой точке $x \in \{x\} \implies \exists f(x) \equiv \lim\limits_{n \to \infty} f_n(x)$ В нер-ве $|f_{n+p}(x) - f_n(x)| < \varepsilon\ \forall p \in \mathcal{N}$ перейдем к lim при $p \to \infty \implies |f(x) - f_n(x)| \le \varepsilon$. По определению это означает $\{f_n(x)\} \rightrightarrows f(x)$ ■

Критерий Коши для функциональных рядов. Функциональный ряд $\sum\limits_{k=1}^{\infty} u_k(x)$ равномерно на множестве $\{x\}$ сходится к некоторой сумме $\iff$ $\forall \varepsilon > 0\ \exists N(\varepsilon) :\ \forall n \geqslant N(\varepsilon),\ \forall p \in \mathbb{N},\ \forall x \in \{x\}$, выполнено $\left|\sum\limits_{k=n+1}^{n+p} u_k(x)\right| < \varepsilon$ ▲ Следует из критерия Коши для функ. последовательностей, так как $\sum\limits_{k=n+1}^{n+p} u_k(x) = S_{n+p}(x) - S_n(x)$ ■

Признак Вейерштрасса. Если функциональный ряд $\sum\limits_{k=1}^{\infty} u_k(x)$ определён на множестве $\{x\}$ пространства E_m и если существует сходящийся числовой ряд $\sum\limits_{k=1}^{\infty} c_k$ такой, что для всех точек x множества $\{x\}$ и для всех номеров k справедливо неравенство $|u_k(x)| \leqslant c_k$, то функциональный ряд $\sum\limits_{k=1}^{\infty} u_k(x)$ сходится равномерно на множестве $\{x\}$.

▲ $\sum c_k \to \Leftrightarrow \forall \varepsilon > 0\ \exists N\ \forall n \ge N\ \forall p \in \mathcal{N} : \sum\limits_{k=n+1}^{n+p} c_n < \varepsilon$
Тогда $\left|\sum\limits_{k=n+1}^{n+p} u_k(x)\right| \le \sum\limits_{k=n+1}^{n+p} |u_k(x)| \le \sum\limits_{k=n+1}^{n+p} c_k < \varepsilon, \forall x \in \{x\}$. В силу критерия Коши теорема доказана. ■

Теорема о почленном переходе к пределу. Если функциональный ряд $\sum\limits_{k=1}^{\infty} u_k(x)$ сходится равномерно на множестве $\{x\}$ к сумме $S(x)$ и у всех членов этого ряда существует в точке x_0 (x_0 — предельная точка множества $\{x\}$) предел $\lim\limits_{x \to x_0} u_k(x) = b_k$, то и сумма ряда $S(x)$ имеет в точке x_0 предел, причём

$$\lim_{x \to x_0} S(x) = \lim_{x \to x_0} \sum_{k=1}^{\infty} u_k(x) = \sum_{k=1}^{\infty} \lim_{x \to x_0} u_k(x) = \sum_{k=1}^{\infty} b_k$$

(или, как говорят, к пределу можно переходить почленно), т.е. символ lim предела и символ $\sum$ суммирования можно переставлять местами.
▲ {Кр. Коши } $\Rightarrow \forall \varepsilon > 0\ \exists N = N(\varepsilon)\ \forall n \ge N\ \forall p \in \mathcal{N} \implies \left|\sum\limits_{k=n+1}^{n+p} u_k(x)\right| < \varepsilon$. Фиксируем n и p и перейдем к пределу при $x \to x_0$ $|b_{n+1} + \dots + b_{n+p}| \le \varepsilon \implies \sum\limits_{k=1}^{\infty} b_k$ сходится.
$\forall x \in U_\delta(x_0) : \left\{S(x) = \sum\limits_{k=1}^{\infty} u_k(x) \forall x \in U_\delta(x_0)\right\} \implies \left|S(x) - \sum\limits_{k=1}^{\infty} b_k\right| \le \left|\sum\limits_{k=1}^{n} u_k(x) - \sum\limits_{k=1}^{n} b_k\right| + \left|\sum\limits_{k=n+1}^{\infty} u_k(x)\right| + \left|\sum\limits_{k=n+1}^{\infty} b_k\right| \forall x \in U_\delta(x_0)$
Оценим слагаемые отдельно: $\forall \varepsilon > 0\ \exists n : \forall x \in \mathcal{E} \implies \left|\sum\limits_{k=n+1}^{\infty} b_k\right| < \frac{\varepsilon}{3}$, так как ряд $\sum\limits_{k=1}^{\infty} b_k$ сходится; $\left|\sum\limits_{k=n+1}^{\infty} u_k(x)\right| < \frac{\varepsilon}{3}$, так как $\sum\limits_{k=1}^{\infty} u_k(x)$ равномерно сходимостся; $\exists \delta > 0 : \forall x \in U_\delta(x_0) \left|\sum\limits_{k=1}^{n} u_k(x) - \sum\limits_{k=1}^{n} b_k\right| < \frac{\varepsilon}{3}$, так как $\lim\limits_{x \to x_0} u_k(x) = b_k$. $\implies \left|S(x) - \sum\limits_{k=1}^{\infty} b_k\right| < \varepsilon \forall x \in U_\delta(x_0) \forall n > N$ ■

Непрерывность предельной функции для ф.п.. $\rceil$ $\forall n \in \mathbb{N} \implies f_n(x) \in C(E)$ и $f_n(x) \rightrightarrows^E f(x)$. Тогда $f(x) \in C(E)$.
▲ $\forall \varepsilon > 0 \exists N(\varepsilon) : \forall n \ge N, \forall x \in E \implies |f_n(x) - f(x)| < \varepsilon$. Выберем $x_0 \in E, \forall x \in U(x_0)$ (ε-окрестность x_0).
👁 $|f(x) - f(x_0)| \le \underbrace{|f(x) - f_N(x)|}_{(1)} + \underbrace{|f_N(x) - f_N(x_0)|}_{(2)} + \underbrace{|f_N(x_0) - f(x_0)|}_{(3)} < \varepsilon$
$(1) < \frac{\varepsilon}{3}, (3) < \frac{\varepsilon}{3}$ в силу сходимости к предельной функции.
$(2) < \frac{\varepsilon}{3}$ в силу непрерывности всех членов. ■

Непрерывность суммы ряда. Если в условиях теоремы о почленном переходе к пределу дополнительно потребовать, чтобы точка x_0 принадлежала множеству $\{x\}$ и чтобы все члены $u_k(x)$ функционального ряда были непрерывны в точке x_0, то и сумма $S(x)$ этого ряда будет непрерывна в точке x_0.
▲ Достаточно применить предыдущую теорему к функциям $f_n(x) = \sum\limits_{k=1}^{n} u_k(x)$ и $f(x) = S(x)$ ■

[И.С. Ломов, *Рукописные лекции по математическому анализу*]

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

Дифференцируемость функции.
$\rceil f(z)$ определена на $U_\delta(z_0)$, $\delta > 0$. Если $\exists \lim\limits_{z \to z_0} \frac{f(z) - f(z_0)}{z - z_0}$, то этот предел называют **производной функции** $f(z)$ в точке z_0 и обозначают $f'(z_0)$.
Положим

$$f(z) = u(x, y) + iv(x, y),\ z = x + iy,\ z_0 = x_0 + iy_0,$$

$$\Delta f = f(z) - f(z_0),\ \Delta f = \Delta u + i\Delta v,\ \Delta z = z - z_0,\ \Delta z = \Delta x + i\Delta y,$$

$$\Delta u = u(x, y) - u(x_0, y_0),\ \Delta v = v(x, y) - v(x_0, y_0),\ \Delta x = x - x_0,\ \Delta y = y - y_0.$$

Функция $f(z)$ называется **дифференцируемой** в точке z_0, если приращение $\Delta f = A\Delta z + \overline{o}(1)\Delta z$, где $\lim\limits_{\Delta z \to 0} \overline{o}(1) = 0$, а A не зависит от Δz.
Условия Коши-Римана: $u'_x = v'_y,\ u'_y = -v'_x$.
Критерий дифференцируемости функции комплексного переменного. Для того, чтобы функция $f(z) = u(x, y) + iv(x, y)$ была дифференцируема в точке $z_0 = x_0 + iy_0$, необходимо и достаточно, чтобы функции $u(x, y)$ и $v(x, y)$ были дифференцируемы в точке (x_0, y_0) и выполнялись условия Коши-Римана.
▲ ($\Longleftarrow$) Предположим, что функции $u(x, y)$ и $v(x, y)$ дифференцируемы в точке (x_0, y_0) и выполнены условия Коши-Римана.
Пусть $u'_x = a = v'_y,\ u'_y = -b = -v'_x$. В силу дифференцируемости функций u и v имеем:

$$\Delta u = a\Delta x - b\Delta y + \overline{o}_1(1)\Delta x + \overline{o}_2(1)\Delta y,$$

$$\Delta v = b\Delta x + a\Delta y + \overline{o}_3(1)\Delta x + \overline{o}_4(1)\Delta y,$$

$$\lim_{\substack{\Delta x \to 0 \\ \Delta y \to 0}} \overline{o}_i(1) = 0,\ i = 1, 2, 3, 4.$$

Отсюда $\Delta f = \Delta u + \Delta iv = a\Delta x - b\Delta y + \overline{o}_1(1)\Delta x + \overline{o}_2(1)\Delta y + ib\Delta x + ia\Delta y + i\overline{o}_3(1)\Delta x + i\overline{o}_4(1)\Delta y =$
$= \Delta x(a + ib) + \Delta y(-b + ia) + \Delta x(\overline{o}_1(1) + i\overline{o}_3(1)) + \Delta y(\overline{o}_2(1) + i\overline{o}_4(1)) =$
$= (a + ib)(\Delta x + \Delta iy) + \Delta x(\overline{o}_1(1) + i\overline{o}_3(1)) + \Delta y(\overline{o}_2(1) + i\overline{o}_4(1))$.
Тем самым

$$\frac{\Delta f}{\Delta z} = a + ib + \frac{\Delta x}{\Delta z}(\overline{o}_1(1) + i\overline{o}_3(1)) + \frac{\Delta y}{\Delta z}(\overline{o}_2(1) + i\overline{o}_4(1)).$$

Так как $|\Delta x| \leqslant |\Delta z|,\ |\Delta y| \leqslant |\Delta z|$, то

$$\lim_{\Delta z \to 0} \frac{\Delta f}{\Delta z} = a + ib = f'(z_0) = u'_x + iv'_x = v'_y - iu'_y = u'_x - iu'_y = v'_y + iv'_x$$

.
($\implies$) Пусть $f(z)$ дифференцируема в точке z_0, тогда $\Delta f = \Delta u + \Delta iv = f'(z_0)\Delta z + \overline{o}(1)\Delta z = f'(z_0)(\Delta x + i\Delta y) + \overline{o}(1)(\Delta x + i\Delta y)$, $\lim\limits_{\Delta z \to 0} \overline{o}(1) = 0$
Пусть $f'(z_0) = a + ib,\ \overline{o}(1) = \alpha_1 + i\alpha_2$, тогда:
$\Delta u + \Delta iv = (a + ib)(\Delta x + i\Delta y) + (\alpha_1 + i\alpha_2)(\Delta x + i\Delta y) = (a\Delta x - b\Delta y) + i(a\Delta y + b\Delta x) + (\alpha_1\Delta x - \alpha_2\Delta y) + i(\alpha_1\Delta y + \alpha_2\Delta x)$
Отдельно для действительной и мнимой частей:

$$\Delta u = a\Delta x - b\Delta y + \alpha_1\Delta x - \alpha_2\Delta y$$

$$\Delta v = b\Delta x + a\Delta y + \alpha_1\Delta y + \alpha_2\Delta x$$

Отсюда, в силу того, что $\lim\limits_{\substack{\Delta x \to 0 \\ \Delta y \to 0}} \alpha_1 = 0,\ \lim\limits_{\substack{\Delta x \to 0 \\ \Delta y \to 0}} \alpha_2 = 0$, следует, что функции u, v дифференцируемы в точке (x_0, y_0) и удовлетворяют условиям Коши-Римана. ■
Аналитическая функция.
Достаточное условие существования $f'(z)$. Если $f(z) = u(x, y) + iv(x, y)$ и функции $u(x, y),\ v(x, y)$ в области D имеют непрерывные частные производные первого порядка и выполняются условия Коши-Римана, то функция $f(z)$ дифференцируема в области D и $f'(z) \in C(D)$.
Функция $f(z)$ называется **аналитической** в области D, если она в каждой точке области D имеет производную $f'(z) \in C(D)$.
Свойства аналитических функций:

1. Аналитическая в области D функция непрерывна в D.
2. Сумма и произведение аналитических функций являются аналитическими функциями. Частное $\frac{f(z)}{g(z)}$ двух аналитических функций является аналитической всюду, где $g(z) \neq 0$.
3. Если $f(z), g(w)$ - аналитические функции, то сложная функция $t = g(f(z))$ является аналитической функцией переменного z.
4. Если функция $w = f(z)$ аналитична в области D, и в окрестности точки $z_0 \in D, f'(z) \neq 0$, то в окрестности точки $w_0 = f(z_0) \in W$ определена обратная функция $z = g(w)$, являющаяся аналитической функцией переменного w. При этом $f'(z_0) = \frac{1}{g'(z_0)}$.

Пусть $f(z)$ - комплекснозначная функция комплексного переменного z, $u(x, y) = Ref(z), v(x, y) = Imf(z), z = x + iy$. Интегралом по кривой L, лежащей в области определения f, называется $\int_L f(z)dz = \int_L (u(x, y)dx - v(x, y)dy) + i\int_L (u(x, y)dy + v(x, y)dx)$.
Теорема. Пусть заданная в односвязной области D функция $f(z)$ - однозначная аналитическая функция. Тогда интеграл от $f(z)$ по любому замкнутому контуру L, лежащему в D, равен нулю: $\oint_I f(z)dz = 0$.
Теорема. Пусть заданная в односвязной области D функция $f(z)$ непрерывна, а интеграл от $f(z)$ по любому замкнутому контуру L, лежащему в D, равен нулю. Тогда $f(z)$ является аналитической функцией в области D.

osn 8. Степенные ряды в действительной и комплексной области. 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:

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

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 $|x| < \dfrac{1}{L}$, и расходится для значений x, удовлетворяющих неравенству $|x| > \dfrac{1}{L}$.
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 $|x| < \dfrac{1}{L}$, и расходится при $|x| > \dfrac{1}{L}$.

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

   В силу свойств верхнего предела все элементы $\sqrt[n]{|a_n|}$, начиная с некоторого номера n, удовлетворяют неравенству $\sqrt[n]{|a_n|} < L + \dfrac{\varepsilon}{2}$.

   Таким образом, начиная с этого номера n, справедливо неравенство $\sqrt[n]{|a_n x^n|} = |x|\sqrt[n]{|a_n|} < \dfrac{L+\frac{\varepsilon}{2}}{L+\varepsilon} < 1$, т. е. ряд абсолютно сходится по признаку Коши.

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

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

   Таким образом, начиная с этого номера k, справедливо неравенство $\sqrt[n_k]{|a_{n_k} x^{n_k}|} = |x|\sqrt[n_k]{|a_{n_k}|} > \dfrac{L-\varepsilon}{L-\varepsilon} = 1$, или $|a_{n_k} x^{n_k}| > 1$, 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 $\dfrac{1}{2|x|}$ найдется номер, начиная с которого $\sqrt[n]{|a_n|} < \dfrac{1}{2|x|}$. Стало быть, начиная с указанного номера, $\sqrt[n]{|a_n x^n|} = |x^n|\sqrt[n]{|a_n|} < \dfrac{1}{2} < 1$, 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

$$R = \frac{1}{\overline{\lim\limits_{n\to\infty}} \sqrt[n]{|a_n|}}$$

(в случае, когда $\overline{\lim\limits_{n\to\infty}} \sqrt[n]{|a_n|} = 0,\ R = \infty$)

▲ Очевидно из предыдущей теоремы ■

Для случая комплексного пространства:

Ряд вида $\sum\limits_{n=0}^{\infty} a_n (z - z_0)^n$ 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. $\overline{\lim\limits_{n\to\infty}} \sqrt[n]{|a_n|} = \infty$), то ряд $\sum\limits_{n=0}^{\infty} a_n (z - z_0)^n$ сходится только в точке z_0.

Если $R = \infty$ (т. е. $\overline{\lim\limits_{n\to\infty}} \sqrt[n]{|a_n|} = 0$), 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.

❑ $\varphi(t), \psi(t)$ 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.

❑ $\varphi(t), \psi(t) \in C[\{t\}]$. Уравнения $x = \varphi(t), y = \psi(t)$ задают параметрически кривую L, если $\exists$ 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.

Спрямляемая кривая — кривая, имеющая конечную длину.

Длина кривой – это предел последовательности длин ломаных, вписанных в эту линию, при условии, что длина наибольшего звена $\to 0$.

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

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

👁 произвольную спрямляемую кривую L на плоскости Oxy, не имеющую точек самопересечения и самоналегания, незамкнутую, ограниченную точками $A,\ B$, описывающуюся параметрическими ур-ями:

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

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

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

$\Delta l_k = |\smile M_{k-1}M_k|$ (длина дуги), $\Delta \equiv \max\limits_{1\leqslant k\leqslant n} \Delta l_k$.

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

$\Delta x_k = x_k - x_{k-1},\ x_k = \varphi(t_k),\ \Delta y_k = y_k - y_{k-1},\ y_k = \psi(t_k)$

👁 три интегральные суммы:

1. $\sigma_1 = \sum\limits_{k=1}^{n} f(N_k)\Delta l_k$
2. $\sigma_2 = \sum\limits_{k=1}^{n} P(N_k)\Delta x_k$
3. $\sigma_3 = \sum\limits_{k=1}^{n} Q(N_k)\Delta y_k$

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

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

$$I_1 = \lim_{\Delta\to 0} \sigma_1 = \int\limits_L f(x, y)dl = \int\limits_a^b f(\varphi(t), \psi(t))\sqrt{\varphi_t'^2(t) + \psi_t'^2(t)}dt$$

Если существуют пределы $I_2,\ I_3$ интегральных сумм $\sigma_2,\ \sigma_3$ при $\Delta \to 0$, то они называются **криволинейными интегралами 2** рода от функций $P,\ Q$ по кривой AB.

$$I_2 = \lim_{\Delta\to 0} \sigma_2 = \int\limits_{\smile AB} P(x, y)dx = \int\limits_{\smile AB} P(M)dx$$

$$I_3 = \lim_{\Delta\to 0} \sigma_3 = \int\limits_{\smile AB} Q(x, y)dy = \int\limits_{\smile AB} Q(M)dy$$

$$I_2 + I_3 = \int\limits_{\smile AB} P(x, y)dx + \int\limits_{\smile AB} Q(x, y)dy = \int\limits_{\smile AB} P(x, y)dx + Q(x, y)dy$$

– **общий криволинейный интеграл 2 рода**.

Из определения криволинейных интегралов следует, что:

1. криволинейный интеграл первого рода не зависит от того, в каком направлении пробегает кривая L, а для криволинейного интеграла второго рода изменение направления кривой ведёт к изменению знака, т.е. $\int\limits_{\smile AB} P(x, y)dx + Q(x, y)dy = -\int\limits_{\smile BA} P(x, y)dx + Q(x, y)dy$
2. физически криволинейный интеграл первого рода представляет собой массу кривой L, линейная плотность которой равна $f(x, y)$; общий линейный интеграл второго рода физически представляет собой работу по перемещению материальной точки A в точку B вдоль кривой L под действием силы, имеющей составляющие $P(x, y)$ и $Q(x, y)$.

Во второй части билета какая-то херня. Закоментированны

совершенно раидомные куски ибо не влезает. Надо что-то придумать.

❑ t — единичный вектор касательной к кривой C, согласованной с k, т.е. положительное направление обхода кривой C совпадает в точке приложения вектора t с направлением этого вектора, и если смотреть с конца нормали k, то контур C ориентирован положительно (Его обход осуществляется против часовой стрелки). Говорят, что ориентация кривой C согласована с нормалью «по правилу штопора».

Опр. Векторным полем в $\mathcal{R}^3$ называется векторная функция, определенная в $\mathcal{R}^3$ (или на $D \subset \mathcal{R}^3$): $\overline{f}(M) : \mathcal{R}^3 \to \mathcal{R}^3$

Опр. Ротором векторного поля $f(M)$ называется $rot\mathcal{A}$

В Орт-Норм Базисе выражение для $rot(f) = rot(f_x\overline{e}_1 + f_y\overline{e}_2 + f_z\overline{e}_3)$: $\left(\dfrac{\partial f_z}{\partial y} - \dfrac{\partial f_y}{\partial z}\right)\overline{e}_1 + \left(\dfrac{\partial f_z}{\partial z} - \dfrac{\partial f_z}{\partial x}\right)\overline{e}_2 + \left(\dfrac{\partial f_y}{\partial x} - \dfrac{\partial f_x}{\partial y}\right)\overline{e}_3$

Формула Грина. ❑ a — векторное поле, дифференцируемое в области D, удовлетворяющей условиям 1 и 2, и такое, что его градиент непрерывен в объединении $D \cup C = \overline{D}$. Тогда справедлива формула

$$\iint\limits_{\overline{D}} (k, rot\ a)d\sigma = \oint\limits_C (a, t)dl$$

Выражение справа обычно называют **циркуляцией векторного поля** a по кривой C, а выражение слева — **потоком векторного поля** $rot\ a$ через область D.

Формулировка Пусть C – положительно ориентированная кусочно-гладкая замкнутая кривая на плоскости, а D – область, ограниченная кривой C. Если $\dfrac{\partial P}{\partial y}, \dfrac{\partial Q}{\partial x} \in \mathcal{C}(D)$, то

$$\oint\limits_C P\,dx + Q\,dy = \iint\limits_D \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}\right) dx\,dy$$

На символе интеграла часто рисуют окружность, чтобы подчеркнуть, что кривая C замкнута.

▲ **Доказательство ф. Грина для простой области** ❑ область D – криволинейная трапеция (область, правильная в направлении OY) : $D = \{(x, y) | a \leq x \leq b, y_1(x) \leq y \leq y_2(x)\}$

Для кривой C, ограничивающей область D зададим направление обхода по часовой стрелке. Тогда:

$$\iint\limits_D \frac{\partial P}{\partial y}\,dx\,dy = \int\limits_a^b dx \int\limits_{y_1(x)}^{y_2(x)} \frac{\partial P}{\partial y}\,dy = \int\limits_a^b (P(x, y_2(x)) - P(x, y_1(x)))\,dx =$$

$$= \int\limits_a^b P(x, y_2(x))\,dx - \int\limits_a^b P(x, y_1(x))\,dx \quad (1)$$

Заметим, что оба полученных интеграла можно заменить криволинейными интегралами: $\int\limits_{C_1} P(x, y)\,dx = -\int\limits_{-C_1} P(x, y)\,dx =$

$$-\int\limits_a^b P(x, y_1(x))\,dx \quad (2) \int\limits_{C_3} P(x, y)\,dx = \int\limits_a^b P(x, y_2(x))\,dx \quad (3)$$ Интеграл по C_1 берётся со знаком «минус», так как согласно ориентации контура C направление обхода данной части – от b до a.

Криволинейные интегралы по C_2 и C_4 будут равны нулю, так как $x = \text{const}$: $\int\limits_{C_2} P(x, y)\,dx = 0 \quad (4) \int\limits_{C_4} P(x, y)\,dx = 0 \quad (5)$

Заменим в (1) интегралы согласно (2) и (3), а также прибавим (4) и (5), равные нулю и поэтому не влияющие на значение выражения:

$$\iint\limits_D \frac{\partial P}{\partial y}\,dx\,dy = \int\limits_{C_1} P(x, y)\,dx + \int\limits_{C_3} P(x, y)\,dx + \int\limits_{C_2} P(x, y)\,dx + \int\limits_{C_4} P(x, y)\,dx$$

Так как обход по часовой стрелке при правой ориентации плоскости является отрицательным направлением, то сумма интегралов в правой части является криволинейным интегралом по замкнутой кривой C в отрицательном направлении: $\iint\limits_D \frac{\partial P}{\partial y}\,dx\,dy = -\int\limits_C P(x, y)\,dx \quad (6)$

Аналогично доказывается формула:

$\iint\limits_D \frac{\partial Q}{\partial x}\,dx\,dy = \int\limits_C Q(x, y)\,dy \quad (7)$ если в качестве области D взять область, правильную в направлении OX.

Сложим (6) и (7): $\int\limits_C P\,dx + Q\,dy = \iint\limits_D \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}\right) dx\,dy$ ■

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 $u_1 + u_2 + \dots + u_k + \dots = \sum\limits_{k=1}^{\infty} u_k$, 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. $S_n = u_1 + u_2 + \dots + u_n = \sum\limits_{k=1}^{n} u_k$.

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

• Ряд $\sum\limits_{k=1}^{\infty} u_k$ называется **абсолютно сходящимся**, если ряд $\sum\limits_{k=1}^{\infty} |u_k|$ также сходится.

• Ряд $\sum\limits_{k=1}^{\infty} u_k$ называется **условно сходящимся**, если сам он сходится, а $\sum\limits_{k=1}^{\infty} |u_k|$ расходится.

Теоремы:

Критерий Коши Ряд $\sum\limits_{k=1}^{n} u_k$ сходится $\iff \forall \varepsilon > 0\ \exists N\ \forall n \geqslant N$

$\forall p \in \mathbb{N} : \left|\sum\limits_{k=n+1}^{n+p} u_k\right| < \varepsilon$.

▲ Обычный критерий Коши для последовательностей, с посл-ю частичных сумм: $|S_{n+p} - S_n| < \varepsilon$. ■

Следствие: **Необходимое условие сходимости ряда:** $\lim\limits_{k\to\infty} u_k = 0$.

▲ Критерий Коши при $p = 1$. ■

Признак Даламбера. Рассмотрим ряд $\sum\limits_{k=1}^{\infty} p_k,\ p_k > 0\ \forall k \geqslant k_0 \geqslant 1$.

П. I: Если для всех номеров k, по крайней мере начиная с некоторого номера, справедливо неравенство $\dfrac{p_{k+1}}{p_k} \leqslant q < 1$ $\left(\dfrac{p_{k+1}}{p_k} \geqslant 1\right)$, то ряд $\sum\limits_{k=1}^{\infty} p_k$ сходится (расходится).

▲ Если $\dfrac{p_{k+1}}{p_k} \geqslant 1$, то $p_{k+1} \geqslant p_k$, а значит $\lim\limits_{k\to\infty} u_k \neq 0$, не выполнено необходимое условие сходимости ряда и ряд расходится.

Рассмотрим ряд из элементов геом. прогрессии:

$\sum\limits_{k=1}^{\infty} q^k = q + q^2 + \dots + \dots = \dfrac{1}{1-q},\ |q| < 1$.

Если $\dfrac{p_{k+1}}{p_k} \leqslant q = \dfrac{q^{k+1}}{q^k}$, то ряд $\sum\limits_{k=1}^{\infty} p_k$ сходится по признаку сравнения, так как сходится ряд $\sum\limits_{k=1}^{\infty} q_k$. ■

П. II: Если $\exists$ предел $\lim\limits_{k\to\infty} \dfrac{p_{k+1}}{p_k} = L$, то ряд сходится при $L < 1$ и расходится при $L > 1$ (для $L = 1$ признак не работает).

▲ $\forall \varepsilon > 0\ \exists N\ \forall k \geq N : L - \varepsilon < \dfrac{p_{k+1}}{p_k} < L + \varepsilon$. Выберем $\varepsilon = \dfrac{1}{2}|L - 1|$.

Если $L < 1$, то $\dfrac{p_{k+1}}{p_k} < 0.5L + 0.5 = q < 1$, свели к 1 части, сходится.

Если $L > 1$, то $\dfrac{p_{k+1}}{p_k} > 0.5L + 0.5 > 1$, свели к 1 части, расходится. ■

Интегральный признак Коши-Маклорена. Пусть при $x \geqslant 1$ функция $f(x) \geq 0$ и не возрастает. Тогда ряд $\sum\limits_{k=1}^{\infty} f(k)$ сходится или расходится одновременно с несобственным интегралом $\int\limits_1^{\infty} f(x)dx$.

▲ $\forall k \in \mathbb{N}\ \forall x \in [k, k+1]$, то $f(k) \geq f(x) \geq f(k+1) \implies f(k) \geq \int\limits_k^{k+1} f(x)dx \geq f(k+1),\ k = 1, \dots, n-1,\ (n \geq 2)$

$f(1) + f(2) + \dots + f(n-1) \geq \int\limits_1^n f(x)dx \geq f(2) + \dots + f(n)$

$S_n - p_1 \leq \int\limits_1^n f(x)dx \leq S_{n-1}$

Если $\int\limits_1^{+\infty} f(x)dx$ сходится, то $\int\limits_1^n f(x)dx \leq M \implies S_n \leq M + p_1 \implies$ сходится

Если $\int\limits_1^{+\infty} f(x)dx$ расходится, то $\{f(x) \geq 0\} \int\limits_1^n f(x)dx \to +\infty \implies S_{n-1} \to +\infty \implies$ расходится ■

Признак Лейбница. Пусть последовательность $\{u_k\},\ u_k > 0\ \forall k \in \mathbb{N}$ является невозрастающей и бесконечно малой. Тогда знакочередующийся ряд $\sum\limits_{k=1}^{\infty} (-1)^k u_k$ сходится.

▲ $S_{2n} = (u_1 - u_2) + (u_3 - u_4) + \dots + (u_{2n-1} - u_{2n}) = (> 0) + (> 0) + \dots + (> 0)$. Поэтому в силу невозрастания последовательности $\{u_k\}$ последовательность $\{S_{2n}\}$ не убывает. С другой стороны, $S_{2n} = u_1 - (u_2 - u_3) - \dots - (u_{2n-2} - u_{2n-1}) - u_{2n} = u_1 - (> 0) - \dots - (> 0) - u_{2n}$. Поэтому в силу невозрастания последовательности $\{u_k\}$ и того, что $u_{2n} \geqslant 0$, последовательность $\{S_{2n}\}$ ограничена сверху числом u_1. Следовательно, $\{S_{2n}\}$ сходится к некоторому числу S. Но из того, что $S_{2n-1} = S_{2n} - u_{2n}$ и $\lim\limits_{n\to\infty} u_{2n} = 0$ (из необх. условий сходимости), вытекает сходимость при $n \to \infty$ последовательности S_{2n-1} к тому же S. ■

[В. А. Ильин, *Математический анализ, продолжение курса*, page 7-22]

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

Производной функции $f(x)$ в точке x_0 называется предел при $\Delta x \to 0$ разностного отношения (если этот предел существует): $f'(x_0) = \lim\limits_{\Delta x\to 0} \dfrac{\Delta y}{\Delta x} = \lim\limits_{\Delta x\to 0} \dfrac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}$ ($x_0 + \Delta x \in$ 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, а $\omega(\Delta x) = o(\Delta x)$ при $\Delta x \to 0$

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

$$\Delta u = A_1\Delta x_1 + \dots + A_m\Delta x_m + \alpha_1\Delta x_1 + \dots + \alpha_m\Delta x_m$$

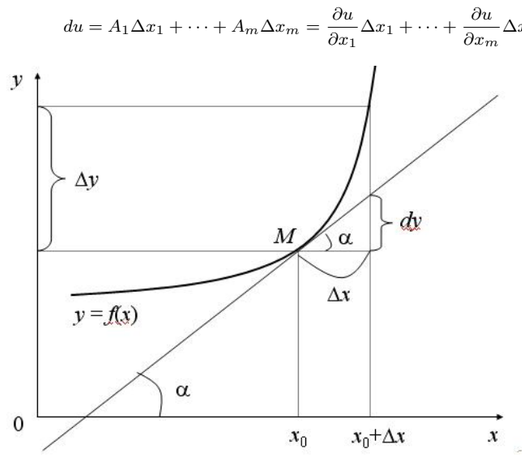
где $A_1, \dots, A_m$ — некоторые не зависящие от $\Delta x_1, \dots, \Delta x_m$ числа, а $\alpha_1, \dots, \alpha_m$ — бесконечно малые при $\Delta x_1 \to 0, \dots, \Delta x_m \to 0$ функции, равные 0 при $\Delta x_1 = \dots = \Delta x_m = 0$

Частная производная функции $f(x_1, \dots, x_m)$ по переменной x_i — это предел отношения приращения функции по x_i к приращению этой переменной, при стремлении этого приращения к нулю:

$$\frac{\partial f}{\partial x_i} = \lim_{\Delta x\to 0} \frac{f(x_1, \dots, x_i + \Delta x, \dots, x_m) - f(x_1, \dots, x_i, \dots, x_m)}{\Delta x}$$

Дифференциалом du дифференцируемой в $M(x_1, \dots, x_m)$ функции $u = f(x_1, \dots, x_m)$ называется главная линейная относительно приращений аргументов часть приращения этой функции в точке M.

$$du = A_1\Delta x_1 + \dots + A_m\Delta x_m = \frac{\partial u}{\partial x_1}\Delta x_1 + \dots + \frac{\partial u}{\partial x_m}\Delta x_m$$



Критерий дифференцируемости для функции одной переменной. Функция одной переменной $f(x)$ дифференцируема в точке $x_0 \iff$ она имеет в этой точке конечную производную.

▲ ($\implies$) $\Delta y = A\Delta x + \alpha(\Delta x)\Delta x \Rightarrow \{\exists \Delta x \neq 0\} \Rightarrow \dfrac{\Delta y}{\Delta x} = A + \alpha(\Delta x) \Rightarrow \lim\limits_{\Delta x\to 0} \dfrac{\Delta y}{\Delta x} = A \Rightarrow f'(x) = A$

($\impliedby$) $\exists f'(x) < \infty . \exists \alpha(\Delta x) = \dfrac{\Delta y}{\Delta x} - f'(x)$, домножим на Δx : $\Delta y = f'(x)\Delta x + \alpha(\Delta x)\Delta x$ ■

Необходимое условие дифференцируемости для функций нескольких переменных. Если $u = f(x_1, \dots, x_m)$ дифференцируема в $M(x_1, \dots, x_m)$, то в этой точке $\exists$ частные производные по всем аргументам, причём $\dfrac{\partial u}{\partial x_i} = A_i$, где A_i определяются из условия дифференцируемости функции.

▲ Из условия дифференцируемости вытекает, что частные приращения $\Delta u_i = A_i\Delta x_i + \alpha_i\Delta x_i \Rightarrow \dfrac{\Delta u_i}{\Delta x_i} = A_i + \alpha_i \Rightarrow \{\alpha_i \to 0 \text{ при } \Delta x_i \to 0\}$

$\Rightarrow \lim\limits_{\Delta x_i\to 0} \dfrac{\Delta u_i}{\Delta x_i} = A_i$ ■

Условие не является достаточным, пример: $f(x, y) = \sqrt{|xy|}$ – $\exists$ ч.п. $f'_x = \dfrac{1}{2}\sqrt{\dfrac{y}{x}}, f'_y = \dfrac{1}{2}\sqrt{\dfrac{x}{y}}$, но не дифф. в т. 0.

Достаточное условие дифференцируемости функций нескольких переменных. Если $u = f(x_1, \dots, x_m)$ имеет частные производные по всем аргументам в некоторой окрестности точки $M_0(x_1^0, \dots, x_m^0)$, причём все эти частные производные непрерывны в точке M_0, то функция дифференцируема в точке M_0.

▲ Для функции двух переменных $u = f(x, y)$: ❑ ч.п. f'_x и f'_y $\exists$ в окр-ти точки $M_0(x_0, y_0)$ и непрерывны в этой точке, ❑ $M(x_0 + \Delta x, y_0 + \Delta y)$ принадлежит указанной окрестности.

$\Delta u = f(x_0 + \Delta x, y_0 + \Delta y) - f(x_0, y_0) = [f(x_0 + \Delta x, y_0 + \Delta y) - f(x_0, y_0 + \Delta y)] + [f(x_0, y_0 + \Delta y) - f(x_0, y_0)]$

$[f(x_0 + \Delta x, y_0 + \Delta y) - f(x_0, y_0 + \Delta y)]$ – приращение ф-ии одной переменной на сегменте $[x_0, x_0 + \Delta x]$. Т.к. $u = f(x, y)$ имеет ч.п., то $f(x, y_0 + \Delta y)$ дифф-ма и ее производная по $x = f'_x$. Применим к указанному приращению формулу Лагранжа:

$[f(x_0 + \Delta x, y_0 + \Delta y) - f(x_0, y_0 + \Delta y)] = f'_x(x_0 + \theta_1\Delta x, y_0 + \Delta y)\Delta x,\ \theta_1 \in (0, 1)$.

Аналогично $[f(x_0, y_0 + \Delta y) - f(x_0, y_0)] = f'_y(x_0, y_0 + \theta_2\Delta y)\Delta y,\ \theta_2 \in (0, 1)$.

f'_x и f'_y непр. в т. $M_0 \Rightarrow f'_x(x_0 + \theta_1\Delta x, y_0 + \Delta y) = f'_x(x_0, y_0) + \alpha$, $f'_y(x_0, y_0 + \theta_2\Delta y)\Delta y = f'_y(x_0, y_0) + \beta$, где α и β – беск. малые при $\Delta x \to 0,\ \Delta y \to 0$ ф-ии.

Отсюда: $\Delta u = f'_x(x_0, y_0)\Delta x + f'_y(x_0, y_0)\Delta y + \alpha\Delta x + \beta\Delta y \Rightarrow u = f(x, y)$ – дифф-ма в точке M_0.

Для ф-ии m переменных $u = f(x_1, \dots, x_m)$ аналогично, представив Δu в виде:

$\Delta u = f(x_1^0 + \Delta x_1, \dots, x_m^0 + \Delta x_m) - f(x_1^0, \dots, x_m^0) = \sum\limits_{i=1}^{m} [f(x_1^0, \dots, x_i^0 + \Delta x_i, \dots, x_m^0 + \Delta x_m) - f(x_1^0, \dots, x_i^0, x_{i+1}^0 + \Delta x_{i+1}, \dots, x_m^0 + \Delta x_m)]$ ■

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.

$$\psi_1, \psi_2, \dots, \psi_n, \dots \tag{2}$$

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 ортонормированная система элементов $\{\psi_k\}$. Рассмотрим какой угодно элемент f пространства E.
Назовём **рядом Фурье** элемента f по ортонормированной системе $\{\psi_k\}$ ряд вида

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

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

$S_n = \sum\limits_{k=1}^{n} f_k\psi_k$ называется n-й **частичной суммой ряда Фурье**.
Рассмотрим наряду с n-й частичной суммой произвольную линейную комбинацию первых n элементов ортонормированной системы $\{\psi_k\}$: $\sum\limits_{k=1}^{n} C_k\psi_k$ с 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: $\min\limits_{\forall\{C_j\}\in\mathbb{R}} \|f - \sum\limits_{k=1}^{n} C_k\psi_k\|$

Будем минимизировать квадрат нормы:
$\|f - \sum\limits_{k=1}^{n} C_k\psi_k\|^2 = \left\langle f - \sum\limits_{k=1}^{n} C_k\psi_k,\ f - \sum\limits_{k=1}^{n} C_k\psi_k \right\rangle = \langle f,f\rangle -$
$2\sum\limits_{k=1}^{n} C_k\langle f,\psi_k\rangle + \sum\limits_{k=1}^{n} C_k^2 = \|f\|^2 + \sum\limits_{k=1}^{n}(C_k^2 - 2C_kf_k) = \left\{\pm\sum\limits_{k=1}^{n} f_k^2\right\} =$
$\|f\|^2 - \sum\limits_{k=1}^{n} f_k^2 + \sum\limits_{k=1}^{n}(C_k - f_k)^2,$
минимум достигается при $C_k = f_k,\ k = 1,2,\dots,n$. Таким образом, доказана следующая теорема:
Теорема. Среди всевозможных линейных комбинаций элементов ортонормированной системы $\{\psi_k\}$ евклидова пространства наименьшее отклонение от произвольного элемента f из пространства имеет n-я частичная сумма ряда Фурье элемента f по системе $\{\psi_k\}$.
Следствие 1. $\forall$ элемента f данного евклидова пространства, $\forall$ ортонормированной системы $\{\psi_k\}$ при произвольном выборе постоянных C_k и $\forall n$ справедливо неравенство

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

▲ $\|f - \sum\limits_{k=1}^{n} C_k\psi_k\|^2 = \|f\|^2 - \sum\limits_{k=1}^{n} f_k^2 + \sum\limits_{k=1}^{n}(C_k - f_k)^2 \geq \|f\|^2 - \sum\limits_{k=1}^{n} f_k^2$ ■
Следствие 2 (тождество Бесселя). $\forall$ элемента f данного евклидова пространства, $\forall$ ортонормированной системы $\{\psi_k\}$ и $\forall n$ справедливо равенство

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

▲ Подставить $C_k = f_k$ в $\|f - \sum\limits_{k=1}^{n} C_k\psi_k\|^2 = \|f\|^2 - \sum\limits_{k=1}^{n} f_k^2 + \sum\limits_{k=1}^{n}(C_k - f_k)^2$
■
Неравенство Бесселя. $\forall$ элемента f данного евклидова пространства, $\forall$ ортонормированной системы $\{\psi_k\}$ выполняется неравенство Бесселя:

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

▲ Из тождества Бесселя: $\|\sum\limits_{k=1}^{n} f_k\psi_k - f\|^2 \geq 0 \implies \|f\|^2 - \sum\limits_{k=1}^{n} f_k^2 \geq 0 \implies \|f\|^2 \geq \sum\limits_{k=1}^{n} f_k^2$ ■
Ортонормированная 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 ортонормированная 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 произвольное $\varepsilon > 0$. 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 $\|f\|^2 - \sum\limits_{k=1}^{n} f_k^2 < \varepsilon$.
$\forall m > n$ это неравенство будет тем более справедливо, так как при возрастании номера n сумма, стоящая в левой части может только возрасти. 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 **афинная 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 **осями координат**. (Обозначается как O_{xyz}). Если вектора e_i взаимно перпендикулярны, то задана **прямоугольная система координат**.

Пусть Oxy — афинная 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:

$$F(x,y) = a_{11}x^2 + 2a_{12}xy + a_{22}y^2 + 2a_{13}x + 2a_{23}y + a_{33} = 0,$$
$$a_{11}^2 + a_{21}^2 + a_{22}^2 \neq 0.$$

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:

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

Тогда уравнение примет вид:

$$F(x,y) = X^TAX + 2b^TX + a_{33} = 0,\ A = A^T,\ A \neq \mathcal{O}.$$

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. $\lambda_1x^2 + \lambda_2y^2 + a_0 = 0$, где $\lambda_1\lambda_2 \neq 0$
2. $\lambda_2y^2 + 2b_0x = 0$, где $\lambda_2b_0 \neq 0$
3. $\lambda_2y^2 + c_0 = 0$, где $\lambda_2 \neq 0$

Эти уранения 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

$$x'' = x' + \frac{a'_{13}}{a'_{11}},\ y'' = y' + \frac{a'_{23}}{a'_{22}},\ a'_{33} = a_{33} - \frac{{a'_{13}}^2}{a'_{11}} - \frac{{a'_{23}}^2}{a'_{22}}$$
$$a'_{11}x''^2 + a'_{22}y''^2 + a'_{33} = 0$$

Все промежуточные и окончательные системы координат оставались прямоугольными, т.к. преобразования базиса с помощью ортогональной матрицы перехода сохраняют свойства ортонормированности. ■
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. $\frac{x^2}{a^2} + \frac{y^2}{b^2} = \pm 1$ — эллипс (мнимый эллипс);
2. $\frac{x^2}{a^2} \pm \frac{y^2}{b^2} = 0$ — пара мнимых пересекающихся прямых (пара пересекающихся прямых); Только начало координат удовлетворяет ур-ю мним. пер. прям.
3. $\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1$ — 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 точна не удовлетворяет ур-ю мним. парал. прям.
2. $y^2 = 0$ — пара совпадающих прямых.

Классификация ПОВЕРХНОСТЕЙ второго порядка
Под **общим уравнением алгебраической поверхности** второго порядка в системе координат $Oxyz$ 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 не все коэффициенты a_{ij} равны нулю, $a_{ij} = a_{ji}$
Введем обозначения:

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

Тогда уравнение примет вид:

$$F(x,y) = X^TAX + 2b^TX + c = 0,\ A \neq \mathcal{O},\ A = A^T.$$

Теорема. С помощью ортогонального преобразования координат (т.е. простым вращением и простым отражением) и параллельного перноса уравнение можно привести к одному из следующих типов:

1. $\lambda_1x^2 + \lambda_2y^2 + \lambda_3z^2 + a_0 = 0,\ \lambda_1\lambda_2\lambda_3 \neq 0$
2. $\lambda_1x^2 + \lambda_2y^2 + b_0z = 0,\ \lambda_1\lambda_2b_0 \neq 0$
3. $\lambda_1x^2 + \lambda_2y^2 + c_0 = 0,\ \lambda_1\lambda_2 \neq 0$
4. $\lambda_2y^2 + p_0x = 0,\ \lambda_1p_o \neq 0$
5. $\lambda_2y^2 + q = 0,\ \lambda_1 \neq 0$

Теорема. Для любой алгебраической поверхности второго порядка существует прямоугольная декартова система координат, в которой уравнение этой поверхности имеет канонический вид:
I тип:

1. $\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = \pm 1$ — эллипсоид (мнимый эллипсоид); Ни одна точка пространства не удовлетворяет ур-ю мним. эллипс.
2. $\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 0$ — вырожденный эллипсоид; Удовлетворяет только начало координат.
3. $\frac{x^2}{a^2} + \frac{y^2}{b^2} - \frac{z^2}{c^2} = \pm 1$ — однополостный гиперболоид (двухполостный гиперболоид);
4. $\frac{x^2}{a^2} + \frac{y^2}{b^2} - \frac{z^2}{c^2} = 0$ — конус;

II тип: $2Z = \frac{x^2}{a^2} \pm \frac{y^2}{b^2}$ — эллиптический параболоид (гиперболический параболоид);
III тип:

1. $\frac{x^2}{a^2} + \frac{y^2}{b^2} = \pm 1$ — эллиптический цилиндр (мнимый эллиптический цилиндр); Ни одна точка пространства не удовлетворяет ур-ю мним. эл. цил.
2. $\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1$ — гиперболический цилиндр;
3. $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 0$ — пара мнимых пересекающихся плоскостей;
4. $\frac{x^2}{a^2} - \frac{y^2}{b^2} = 0$ — пара пересекающихся плоскостей;

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.

[Г. Д. Ким, *Линейная алгебра и аналитическая геометрия*, page 192-200, 329-341]

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

Полем называется множество F с введенными на нем алгебраическими операциями сложения и умножения, а также если выполнены следующие аксиомы:

- Коммутативность сложения: $\forall a,b \in F\ a+b = b+a$
- Ассоциативность сложения: $\forall a,b,c \in F\ (a+b)+c = a+(b+c)$
- Существование нулевого элемента: $\exists 0 \in F : \forall a \in F\ a+0 = 0$
- Существование противоположного элемента: $\forall a \in F \exists(-a) \in F : a+(-a) = 0$
- Коммутативность умножения: $\forall a,b \in F : a*b = b*a$
- Ассоциативность умножения: $\forall a,b,c \in F\ (a*b)*c = a*(b*c)$
- Существование единичного элемента: $\exists e \in F \setminus \{0\} : \forall a \in F\ a*e = a$
- Существование обратного элемента для ненулевых элементов: $(\forall a \in F : a \neq 0) \exists a^{-1} \in F : a*a^{-1} = e$
- Дистрибутивность умножения относительно сложения: $\forall a,b,c \in F\ (a+b)*c = a*c + b*c$

👁 множество V элементов $x,y,z\dots$ и поле P действительных или комплексных чисел. ⊐ в V введены две операции: сложение его элементов и умножение его элементов на числа из P. Т.е $\forall x,y \in V$ определён элемент $z = x+y \in V$, а $\forall x \in V,\ \forall\lambda \in P$ определён элемент $y = \lambda\cdot x \in V$. ⊐ введённые две операции удовлетворяют **следующим аксиомам**:

1. $x+y = y+x$;
2. $(x+y)+z = x+(y+z)$;
3. $\exists\theta \in V$, что $\forall x \in V \implies x+\theta = x$;
4. $\forall x \in V\ \exists(-x) \in V$, что $x+(-x) = \theta$;
5. $1\cdot x = x,\ 1 \in P$;
6. $\lambda\cdot(x+y) = \lambda\cdot x + \lambda\cdot y,\ \lambda \in P$;
7. $(\lambda+\mu)\cdot x = \lambda\cdot x + \mu\cdot x,\ \lambda,\mu \in P$;
8. $(\lambda\mu)\cdot x = \lambda(\mu\cdot x)$.

Тогда V называется **линейным пространством** над полем P.
Если P — поле действительных чисел, то V — **действительное линейное пространство**.
Если P — поле комплексных чисел, то V — **комплексное линейное пространство**.
Максимальное число линейно независимых векторов пространства V называется его **размерностью**. Если размерность пространства V конечна, то оно называется **конечномерным**.
⊐ даны 2 линейных пространства V и W над общим полем P. Отображение $A : V \to W$ называется **линейным отображением (линейным оператором)**, если для $\forall x,y \in V,\ \alpha \in P$ выполнены равенства:

1. $A(x+y) = A(x) + A(y)$;
2. $A(\alpha x) = \alpha A(x)$;

$\mathcal{L}(V,W)$ — множество всех линейных операторов действующих из V в W.
Простейшие свойства.

1. *Линейный оператор переводит нулевой вектор в нулевой вектор*, так как $\mathcal{A}\theta_1 = \mathcal{A}(0x) = 0\mathcal{A}x = \theta_2$ (здесь θ_1,θ_2 - нулевые векторы пространств V и W соответственно)
2. *Линейный оператор сохраняет линейные комбинации*, т.е. переводит линейную комбинацию векторов в линейную комбинацию образов с теми же коэффициентами: $\mathcal{A}\left(\sum\limits_{i=1}^{k}\alpha_ix_i\right) = \sum\limits_{i=1}^{k}\alpha_i\mathcal{A}x_i$
3. *Линейный оператор сохраняет линейную зависимость*, т.е. переводит линейно зависимую систему векторов в линейно зависимую.

Теорема. ⊐ $e_1,e_2,\dots,e_n$ — базис пространства V, а $g_1,g_2,\dots,g_n$ — любые векторы пространства W. Тогда существует единственный линейный оператор $A : V \to W$, который переводит векторы $e_1,e_2,\dots,e_n$ в векторы $g_1,g_2,\dots,g_n$ соответственно.
▲ Строим оператор по правилу: если $x = \sum\limits_{i=1}^{n} x_ie_i \in V$, то $Ax = \sum\limits_{i=1}^{n} x_iAe_i$. Из единственности разложения вектора по базису следует, что правило однозначно определяет образ x, при этом $Ae_i = g_i$. Линейность оператора вытекает из линейности координат. Если B — любой другой оператор, удовлетворяющий условию теоремы, то $Bx = \sum\limits_{i=1}^{n} B(x_ie_i) = \sum\limits_{i=1}^{n} x_ig_i = Ax \implies A$ единственен. ■
Матрица линейного оператора.
⊐ $e_1,e_2,\dots,e_n$ и $f_1,f_2,\dots,f_m$ — базисы конечномерных пространств V и W. Линейный оператор $A : V \to W$ однозначно определяется заданием векторов $Ae_1,\dots,Ae_n$. В свою очередь Ae_i однозначно определяются своими координатами в базисе f:

$$\begin{cases} Ae_1 = a_{11}f_1 + \dots + a_{m1}f_m \\ Ae_2 = a_{12}f_1 + \dots + a_{m2}f_m \\ \dots \\ Ae_n = a_{1n}f_1 + \dots + a_{mn}f_m \end{cases}$$

$A_{fe} = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ & \dots & \\ a_{m1} & \dots & a_{mn} \end{pmatrix}$ называется **матрицей оператора** A в паре базисов e и f.
⊐ V — линейное пространство над полем P. **Нормой** в линейном пространстве V называется отображение $\|\cdot\| : V \to R$, ставящее в соответствие каждому вектору $x \in V$ действительное число $\|x\| \in R$ и удовлетворяет аксиомам: $\forall x,y \in V, \alpha \in P$

1. $\|x\| \geqslant 0$, причём норма равна нулю только если $x = 0$;
2. $\|\alpha x\| = |\alpha|\cdot\|x\|$;
3. $\|x+y\| \leqslant \|x\| + \|y\|$.

Линейное пространство V с заданной на нём нормой $\|\cdot\|$ называется **линейным нормированным пространством.** Число $\|x\|$ называется нормой вектора x.
⊐ V, W — линейные нормированные пространства с нормами $\|\cdot\|_V$ и $\|\cdot\|_W$. ⊐ $\mathcal{L}(V,W)$ — линейное пространство операторов, в котором можно ввести норму со следующими ограничениями $\forall\mathcal{A} \in \mathcal{L}(V,W)$:

- **согласованность** с векторными нормами $\|\cdot\|_V$ и $\|\cdot\|_W$: $\|\mathcal{A}x\|_W \leqslant \|\mathcal{A}\|\cdot\|x\|_V,\ \forall x \in V$.
- **мультипликативность**: $\|\mathcal{AB}\| \leqslant \|\mathcal{A}\|\cdot\|\mathcal{B}\|,\ \forall\mathcal{A},\mathcal{B}$, для которых определено произведение $\mathcal{AB}$.

Теорема. Собственное значение линейного оператора $\mathcal{A} \in \mathcal{L}(V,V)$ не превосходит по абсолютной величине любую его согласованную норму $\|\mathcal{A}\|$.
▲ Если $\mathcal{A}x = \lambda x$, то для любой согласованной нормы оператора имеем $\|\mathcal{A}x\| = |\lambda|\cdot\|x\|$ и $\|\mathcal{A}x\| \leqslant \|\mathcal{A}\|\cdot\|x\|$, откуда следует, что $|\lambda| \leqslant \|\mathcal{A}\|$ ■
Примеры операторных норм:

- $\mu(\mathcal{A}) = \sup\limits_{\|x\|_V=1} \|\mathcal{A}x\|_W$ — норма, **подчинённая** нормам $\|\cdot\|_V$ и $\|\cdot\|_W$, наименьшая из всех согласованных норм.
- $\|\mathcal{A}\|_2 = \sup\limits_{\|x\|_V=1} \sqrt{(\mathcal{A}x,\mathcal{A}x)}$ — **спектральная норма**, равная максимальному сингулярному числу (или максимальному по модулю собственному значению в случае нормального оператора).
- $\|\mathcal{A}\|_1 = \max\limits_{1\leqslant j\leqslant n} \sum\limits_{i=1}^{m} |a_{ij}|$ — **максимальная столбцовая сумма**.
- $\|\mathcal{A}\|_\infty = \max\limits_{1\leqslant i\leqslant m} \sum\limits_{j=1}^{n} |a_{ij}|$ — **максимальная строчная сумма**.
- $\|\mathcal{A}\|_E = \sqrt{\sum\limits_{i,j} |a_{ij}|^2}$ — **евклидова норма оператора.**

[Г. Д. Ким, *Линейная алгебра и аналитическая геометрия*, page 240-241, 352-353]

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

Линейным оператором векторного пространства V над полем P в векторное пространство W над тем же полем P называется оператор $\mathcal{A} \in \mathcal{L}(V,W)$, удовлетворяющий условию линейности для всех $x,y \in V$ и $\alpha \in P$:

$$\mathcal{A}(x+y) = \mathcal{A}x + \mathcal{A}y$$
$$\mathcal{A}(\alpha x) = \alpha\mathcal{A}x$$

Пусть $e = (e_1,\dots,e_n)$ и $f = (f_1,\dots,f_m)$ – базисы пространств V и W. Линейный оператор $\mathcal{A} \in \mathcal{L}(V,W)$ однозначно определяется заданием векторов $\mathcal{A}e_1,\dots,\mathcal{A}e_n$. В свою очередь вектора $\mathcal{A}e_i, i \in 1..n$ своими координатами в базисе f, т.е. коэффициентами разложений

$$\begin{cases} \mathcal{A}e_1 = a_{11}f_1 + a_{21}f_2 + \dots + a_{m1}f_m \\ \dots \\ \mathcal{A}e_n = a_{1n}f_n + a_{2n}f_2 + \dots + a_{mn}f_m \end{cases}$$

Матрица $A_{fe} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$ называется **матрицей линейного оператора** $\mathcal{A}$ в базисе e и f. Матрицы линейного оператора в различных парах базисов эквивалентны.
Матрицы $A,B \in P^{m*n}$ называются **эквивалентными**, если существуют м-цы $D \in P^{m*m}, |D| \neq 0$ и $Q \in P^{n*n}, |Q| \neq 0$, такие что $B = DAQ$.
Квадратные матрицы $A,B \in P^{n*n}$ называются **подобными**, если существует м-ца $D \in P^{n*n}, |D| \neq 0$, такая что $B = D^{-1}AD$.
Собственные числа и собственные векторы.
Пусть V – линейное пространство над полем P и $\mathcal{A} \in \mathcal{L}(V,V)$.
Вектор $x : x \neq \theta, x \in V$ называется **собственным вектором** оператора $\mathcal{A} = \mathcal{L}(V,V)$, если $\exists\lambda \in P$, такой что :

$$\mathcal{A}x = \lambda x$$

λ называется **собственным значением** оператора $\mathcal{A}$, соответствующим собственному вектору x.
Спектр оператора $\mathcal{A}$ – это множество всех собственных значений $\mathcal{A}$.
Теорема: Собственные векторы, отвечающие различным собственным значениям, линейно независимы.
▲ По индукции, для $n = 1$ верно, т.к. с.в. $\neq \theta$ по определению. Пусть верно для из $n-1$ векторов, докажем для n. От противного: пусть $x_1,\dots,x_n$ — собственные векторы, отвечающие различным собственным значениям $\lambda_1,\dots,\lambda_n$, — линейно зависимы. Тогда существует нетривиальная линейная комбинация этих векторов:

$$\alpha_1x_1 + \dots + \alpha_nx_n = \theta$$

Подействуем на неё оператором A : $\alpha_1\lambda_1x_1 + \dots + \alpha_n\lambda_nx_n = \theta$
Домножим линейную комбинацию на λ_n : $\alpha_1\lambda_nx_1 + \dots + \alpha_n\lambda_nx_n = \theta$
Вычтем первое равенство из второго, получим:

$$\alpha_1(\lambda_n - \lambda_1)x_1 + \dots + \alpha_{n-1}(\lambda_n - \lambda_{n-1})x_{n-1} = \theta$$

Получили нетривиальную линейную комбинацию линейно независимых векторов, значит предположение неверно и теорема верна. ■
Следствие: Линейный оператор, действующий в n-мерном пространстве, не может иметь более n различных собственных значений.
Характеристический многочлен.
$f(\lambda) = |A - \lambda I|$ — **характеристический многочлен** матрицы A.
Уравнение $det(\mathcal{A} - \lambda I) = 0$ — **характеристическое уравнение** оператора $\mathcal{A}$.
Теорема: Характеристические многочлены подобных матриц совпадают, т.е. $A = Q^{-1}BQ \implies |A - \lambda I| = |B - \lambda I|$.
▲ $|A - \lambda I| = |Q^{-1}BQ - \lambda I| = |Q^{-1}BQ - Q^{-1}\lambda IQ| = |Q^{-1}(B - \lambda I)Q| = |B - \lambda I|$.
Т.е. $\forall\lambda$ характеристические многочлены м-ц A и B принимают одинаковые значения. ■
Следствие: Все матрицы одного и того же линейного оператора имеют одинаковые характеристические многочлены.

Образом лин. оператора $\mathcal{A} \in \mathcal{L}(V,W)$ наз-ся мн-во $im\mathcal{A} = \{y \in W | \exists x \in V : \mathcal{A}x = y\}$.
Ядром лин. оператора $\mathcal{A} \in \mathcal{L}(V,W)$ наз-ся мн-во $ker\mathcal{A} = \{x \in V | \mathcal{A}x = 0\}$.
Теорема: λ — собственное значение оператора $\mathcal{A} \iff \lambda$ — корень характеристического уравнения оператора $\mathcal{A}$.
▲ $(\implies)$: $\lambda \in P$ — собственное значение, $\exists x : Ax = \lambda x,\ x \neq 0,\ (A - \lambda I)x = 0 \implies dim(ker(A - \lambda I)) \neq 0 \implies det(A - \lambda I) = 0$.
$(\impliedby)$: $\exists\lambda$ — корень характеристического уравнения $\implies dim(im(A - \lambda I)) \leqslant n-1 \implies dim(ker(A - \lambda I)) \geqslant 1 \implies \exists x \neq \theta : (A - \lambda I)x = 0$. ■

[Г. Д. Ким, *Линейная алгебра и аналитическая геометрия*, page 240-260]

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, а для другого — нет.
Алгоритм **применим** к входным данным, если исполнитель за конечное число шагов остановится и выдаст (какой-то) ответ. В противном случае алгоритм **неприменим** к конкретным входным данным, т.е. он не остановится, либо завершит своё 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 зациклится 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, 👁 два из них: машины Тьюринга и нормальные алгоритмы Маркова.
Машина Тьюринга — гипотетическая машина (из-за использования бесконечной ленты). Автомат может двигаться вдоль ленты и по очереди 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	$, R,$	анализ I симв., удаление
q_2	$, R,$	$, R,$	$, R,$	$a, , !$	запись a справа
q_3	$, R,$	$, R,$	$, R,$	$b, , !$	запись b справа
q_4	$, R,$	$, R,$	$, R,$	$c, , !$	запись c справа

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 одинаковым множествам слов. В случае 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: $A_i \left\{ \begin{array}{c} \to \\ \mapsto \end{array} \right\} B_i$. Каждая пара – формула подстановки для замены подслов в преобразуемом слове. Ищется вхождение слова A_1 в исходное слово. Если нашли, то заменяем 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 применилась завершающая формула, в которой $\mapsto$.
Пример: $A = \{a, b\}$. Преобразовать слово P так, чтобы в его начале оказались все символы a, а в конце – все символы b.

$$\begin{cases} ba \to ab \end{cases}$$

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 одинаковым множествам слов. Машина 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 одного 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 входе запись любого алгоритма A и его конкретные входные данные D, определяет, применим ли A к 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:

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

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 $Z = X * Y$. 👁, самоприменим ли Z.

– Пусть Z самоприменим, тогда ⟨запись Z⟩$Z \to$ ⟨запись Z⟩$X * Y \to$ ⟨DA⟩$Y \to$ Зациклились, предположение неверно.

– Пусть Z несамоприменим, тогда ⟨запись Z⟩$Z \to$ ⟨запись Z⟩$X * Y \to$ ⟨NET⟩$Y \to$ 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.
Ортонормированным базисом называется базис, векторы которого имеют единичную длину и в случае n>1 попарно перпендикулярны.
Ортогональные матрицы и их свойства.

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

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

Q^H – *эрмитова-сопряженная матрица*: $q_{ji} = \overline{q}_{ij}$
Q^T – *транспонированная матрица*: $q_{ij} = q_{ji}$
Свойства ортогональной матрицы:

1. Q — обратима, причём $Q^{-1} = Q^T$;
   ▲ $\left|Q^T\right| = |Q| \implies |Q|^2 = 1 \implies |Q| \neq 0 \implies \exists Q^{-1}, Q^T Q = I \implies$ {домн. справа на Q^{-1} } $\implies Q^T = Q^{-1}$ ■

2. $det(Q) = \pm 1$;
   ▲ $\left|Q^T\right| = |Q| \implies |Q|^2 = 1 \implies |Q| = \pm 1$ ■

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

Линейный оператор $\mathcal{U}$, действующий в унитарном (евклидомов) пространстве, называется **унитарным** (**ортогональным**) оператором, если $\mathcal{U}^*\mathcal{U} = \mathcal{U}\mathcal{U}^* = \mathcal{I}$

• Оператор $\mathcal{U}$ унитарен (ортогонален) $\iff$ в любом ортонормировавнном базисе он имеет унитарную (ортогональную) матрицу.

• Для унитарного (ортогонального) оператора $\mathcal{U}$ справедливы равенства:
$\mathcal{U}^* = \mathcal{U}^{-1}$, $|det\mathcal{U}| = 1$.

• Унитарный (ортогональный) оператор нормален.

Критерий унитарности. В конечномерном унитарном (евклидовом) пространстве $\mathcal{V}$ следующие утверждения равносильны:

• Оператор $\mathcal{U}$ унитарен (ортогонален)

• $\mathcal{U}^*\mathcal{U} = \mathcal{I}$

• $\mathcal{U}\mathcal{U}^* = \mathcal{I}$

• оператор $\mathcal{U}$ изометричен

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

• оператор $\mathcal{U}$ переводит любой ортонормированный базис $\mathcal{V}$ в ортонормированный базис

• оператор $\mathcal{U}$ переводит хотя бы один ортонормированный базис $\mathcal{V}$ в ортонормированный базис

▲

• $(1 \Leftrightarrow 2)$ $(\implies)$ очевидно
$(\impliedby)$ $UU^* = I \implies \exists U^{-1}, UU^* = I$ {домн. слева на U^{-1}1} $\implies U^* = U^{-1} \implies U^*U = I$

• $(1 \Leftrightarrow 3)$ аналогично

• $(3 \Leftrightarrow 4)$ $(\implies)$ $(Ux, Uy) = (x, UU^*y) = (x, y)$
$(\impliedby)$ $(Ux, Uy) = (x, y) \implies (x, UU^*y) = (x, Iy) \implies UU^* = I$

• $(4 \Leftrightarrow 5)$ $(\implies)$ очевидно, т.к. $|x| = \sqrt{(x, x)}$
$(\impliedby)$ а). V – евкл.: $(x, y) = (|x + y|^2 - |x|^2 - |y|^2)/2$ б). V – унит.: $(x, y) = (|x + y|^2 - |x - y|^2 + i\,|x + iy|^2 - i\,|x - iy|^2)/2$

• $(4 \Leftrightarrow 5)$ $(\implies)$ e – о/н $\implies (Ue_i, Ue_j) = (e_i, e_j) = \delta_{ij}$
$(\impliedby)$ Пусть $e_1, \dots, e_n$ – о/н. $\forall x \in V : x = \sum x_i e_i, Ux = \sum x_i Ue_i \implies |x|^2 = \sum |x_i|^2, |Ux|^2 = \sum |x_i|^2 \implies U$ сохр. длину

• $(6 \Leftrightarrow 7)$ $(\implies)$ очевидно
$(\impliedby)$ доказано в предыдущем пункте $(\impliedby)$

■

Примеры ортогональных матриц.

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

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

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

$$\begin{cases} q_{11} = \cos\varphi \\ q_{21} = \sin\varphi \\ q_{12} = -kq_{21} = -k\sin\varphi \\ q_{22} = kq_{11} = k\cos\varphi \end{cases}$$

1. $k = 1 \implies Q = \begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix}$ (поворот).
   При $\varphi = \pi k$ получаются матрицы $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$.
   При $\varphi \neq \pi k$ матрица не диагонализируется, так как у неё нет вещественных собственных значений.

2. $k = -1 \implies Q = \begin{bmatrix} \cos\varphi & \sin\varphi \\ \sin\varphi & -\cos\varphi \end{bmatrix}$ (поворот и отражение).
   В этом случае собственные значения матрицы равны $\pm 1 \implies$ получаются матрицы $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$.

Таким образом, в случае $Q \in \mathbb{R}^{2\times 2}$ для ортогонального оператора в евклидовом пространстве существует ортонормированный базис, в котором он имеет матрицу

либо $\begin{bmatrix} \pm 1 & 0 \\ 0 & \pm 1 \end{bmatrix}$,

либо $\begin{bmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{bmatrix}$, $(\varphi \neq \pi k)$.

Теорема. $\forall$ ортогонального оператора Q в евклидовом пространстве $\exists$ ортонормированный базис e, в котором матрица оператора имеет вид:

$$Q = \begin{bmatrix} 1 & & & & & & & & \\ & \ddots & & & & & & & \\ & & 1 & & & & & & \\ & & & -1 & & & & & \\ & & & & \ddots & & & & \\ & & & & & -1 & & & \\ & & & & & & \cos\varphi_1 & -\sin\varphi_1 & \\ & & & & & & \sin\varphi_1 & \cos\varphi_1 & \\ & & & & & & & & \ddots \end{bmatrix}$$

▲ Доказательство методом математической индукции по размерности пространства $dim(V) = n$.
Для $n = 1,\ n = 2$ теорема верна (выше рассмотрены все возможные случаи).
Пусть теорема верна для $n - 1,\ n - 2$. Докажем для n.
Q действует в вещественном пространстве, а в нём существует либо одномерное, либо двумерное инвариантное подпространство L ($\forall x \in L\ Qx \in L$), для него можно найти базис. По свойствам ортогонального оператора $L^\perp$ (ортогональное подпространство) — инвариантно относительно Q. $dim(L^\perp)$ равна либо $n - 1$, либо $n - 2 \implies$ теорема для него верна. Совокупность базисов для L и $L^\perp$ образует искомый базис с точностью до порядка базисных векторов. ■

[Г. Д. Ким, *Линейная алгебра и аналитическая геометрия*, page 230-233, 292-295]

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

Системой m линейных алгебраических уравнений с n неизвестными называется совокупность отношений:

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

Упорядоченная совокупность чисел $c_1, ..., c_n \in \mathbb{R}$ называется **решением** системы, если при подстановке этих чисел в систему вместо неизвестных $x_1, ..., x_n$ соответственно каждое уравнение обращается в тождество. Две СЛАУ **эквивалентны**, если множества их решений совпадают. СЛАУ **совместна**, если существует хотя бы одно решение. СЛАУ называется **определенной**, если она имеет единственное решение, если имеет больше одного – **неопределенной**
Теорема. СЛАУ с квадратной невырожденной матрицей совместна и имеет единственное решение.
▲ В силу невырожденности матрицы A для нее существует обратная матрица A^{-1}. Вектор $x = A^{-1}b$ – решение. Оно единственно. Если y – другое решение системы, то $Ax \equiv Ay$. Умножив обе части тождества слева на A^{-1}, получим $x = y$ ■
Решение СЛАУ с помощью правила Крамера.
Введём: $A_i = \begin{pmatrix} a_{1,1} & \dots & a_{1,i-1} & b_1 & a_{1,i+1} & \dots & a_{1,n} \\ a_{2,1} & \dots & a_{2,i-1} & b_2 & a_{2,i+1} & \dots & a_{2,n} \\ & & & \dots & & & \\ a_{n,1} & \dots & a_{n,i-1} & b_n & a_{n,i+1} & \dots & a_{n,n} \end{pmatrix}$
Тогда: $x_i = \dfrac{|A_i|}{|A|},\ i = 1, \dots, n$. A^{-1} получается из матрицы A заменой ее i-го столбца столбцом свободных членов.
$B = (A|b)$ — **расширенная** матрица.
Теорема Кронеккера-Капелли. СЛАУ совместна $\iff rgB = rgA$.
▲ $(\implies)$ Пусть СЛАУ совместна $\implies \exists x_1, x_2, \dots, x_n :\ a_1x_1 + a_2x_2 + \dots + a_nx_n = b \implies$ столбец b является линейной комбинацией столбцов $a_1, \dots, a_n \implies rgB = rg(A|b) = rgA$
$(\impliedby)$ Пусть $rgB = rg(A|b) = rgA = r$. Возьмём в матрице A произвольный базисный минор. Так как $rg(A|b) = r$, то он же будет базисным минором матрицы $(A|b)$. Следовательно, последний столбец матрицы $(A|b)$ будет являться линейной комбинацией столбцов матрицы A. Коэффициенты этой комбинации являются решением СЛАУ $Ax = B$, то есть система совместна. ■
Пусть система

$$\begin{cases} a_{11}x_1 + \dots + a_{1r}x_r + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + \dots + a_{2r}x_r + \dots + a_{2n}x_n = b_2 \\ \dots \\ a_{n1}x_1 + \dots + a_{nr}x_r + \dots + a_{nn}x_n = b_n \end{cases}$$

совместна и $rgA = rgB = r$. Будем считать, что базисный минор матрицы A находится в левом верхнем углу:

$$M = \begin{vmatrix} a_{11} & \dots & a_{1r} \\ & \dots & \\ a_{r1} & \dots & a_{rr} \end{vmatrix} \neq 0$$

Рассмотрим **укороченную** систему:

$$\begin{cases} a_{11}x_1 + \dots + a_{1r}x_r + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + \dots + a_{2r}x_r + \dots + a_{2n}x_n = b_2 \\ \dots \\ a_{r1}x_1 + \dots + a_{rr}x_r + \dots + a_{rn}x_n = b_r \end{cases} \quad (3)$$

Теорема. Укороченная система эквивалентна исходной системе.
▲ Обе системы содержат одинаковое число неизвестных. Любое решение *исходной* системы является рещением системы (3). Покажем, что верно и обратное. В расширенной матрице исходной системы первые r строк являются базисными. Следовательно, все остальные строки согласно теореме о базисном миноре будут линейными комбинациями этих строк. Это означает, что каждое уравнение исходной системы, начиная с $(r + 1)$-го, будет линейной комбинацией первых r уравнений этой системы. Следовательно, каждое решение первых r уравнений исходной системы обращает в тождества все последующие уравнения. ■
Запишем систему в виде

$$\begin{cases} a_{11}x_1 + \dots + a_{1r}x_r = b_1 - a_{1,r+1}x_{r+1} - \dots - a_{1n}x_n \\ a_{21}x_1 + \dots + a_{2r}x_r = b_2 - a_{2,r+1}x_{r+1} - \dots - a_{2n}x_n \\ \dots \\ a_{r1}x_1 + \dots + a_{rr}x_r = b_r - a_{r,r+1}x_{r+1} - \dots - a_{rn}x_n \end{cases}$$

Придав свободным членам $x_{r+1}, \dots, x_n$ произвольные значения $c_{r+1}, \dots, c_n$, получим систему уравнений относительно неизвестных $x_1, x_2, \dots, x_r$ с квадратной невырожденной матрицей:

$$\begin{cases} a_{11}x_1 + \dots + a_{1r}x_r = b_1 - a_{1,r+1}c_{r+1} - \dots - a_{1n}c_n \\ a_{21}x_1 + \dots + a_{2r}x_r = b_2 - a_{2,r+1}c_{r+1} - \dots - a_{2n}c_n \\ \dots \\ a_{r1}x_1 + \dots + a_{rr}x_r = b_r - a_{r,r+1}c_{r+1} - \dots - a_{rn}c_n \end{cases} \quad (4)$$

Эта система имеет единственное решение $c_1, c_2, \dots, c_r$. Очевидно, совокупность $c_1, c_2, \dots, c_n$ является решением исходной системы.
Теорема. Придавая свободным неизвестным произвольные значения и вычисляя значения главных неизвестных, из полученной системы можно получить все решения исходной системы.
▲ Пусть $(c_1, \dots, c_r, c_{r+1}, \dots, c_n)$ — произвольное решение (3). Возьмём числа $(c_{r+1}, \dots, c_n)$ в качестве свободных переменных $x_{r+1}, \dots, x_n$ и будем вычислять значения главных неизвестных из системы (4). Так как $(c_1, \dots, c_r, c_{r+1}, \dots, c_n)$ — решение (3), то $(c_1, \dots, c_r)$ — решение системы (4). Так как система (4) имеет единственное решение, то в качестве решения можем получить только $(c_1, \dots, c_r)$. ■
Опр. Система линейных алгебраических уравнений с нулевой правой частью называется **однородной**.
Теорема. СЛАУ с n неизвестными имеет единственное решение $\iff rgB = rgA = n$.
▲ Если $rgA < n$, то среди неизвестных будет хотя бы одно свободное неизвестное. Тогда получим бесконечно много решений. ■
Общее решение СЛАУ Решим полученную систему (3) относительно главных неизвестных: $x_1 = f_1(x_{r+1}, \dots, x_n)$, $\dots$, $x_r = f_r(x_{r+1}, \dots, x_n)$, где $f_1, \dots, f_r$ – однозначно определённые функции. Эти соотношения при произвольных $x_{r+1}, \dots, x_n$ описывают множество всех решений исходной системы и называются **общим решением** системы.
сюда бы пример нахождения общего решения

• Однородная СЛАУ $Ax = 0$ всегда совместна: имеет тривиальное решение $x = \theta$.

• Однородная система с n неизвестными имеет нетривиальное решение $\iff rgA < n$.

• Однородная система $Ax = 0$ с квадратной матрицей A имеет нетривиальное решение $\iff |A| = 0$.

[Г. Д. Ким, *Линейная алгебра и аналитическая геометрия*, page 104-109]

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

Если в пространстве V_3 зафиксированы точка O и базис $\{e_1, e_2, e_3\}$, то говорят что в пространстве задана **афинная 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 **осями координат**. (Обозначается как O_{xyz})
Ненулевой вектор коллинеарный прямой называется ее **направляющим вектором**: $\overline{M_0M} = t\overline{a}$
Два неколлинеарных вектора, параллельных плоскости, называются ее **направляющими векторами**.

Канонические уравнения.

Уравнение прямой. На плоскости в афинной системе координат O_{xy} уравнение прямой l, проходящей через точку $M_0(x_0, y_0)$ с направляющим вектором $\overline{a} = \{l, m\}$ имеет вид:

$$\begin{vmatrix} x - x_0 & y - y_0 \\ l & m \end{vmatrix} = 0 \iff \frac{x - x_0}{l} = \frac{y - y_0}{m}$$

▲ $\sqsupset M(x, y)$ – точка. Тогда $\overline{M_0M} = (x - x_0,\ y - y_0)$. Условие $M_0M = ta$ в силу лин. координат означает, что в определении первая строка линейно выражена через вторую, а это равносильно определителю (выше). Равенство нулю определителя второго порядка равносильно пропорциональности его строк. ■
Уравнение плоскости. В пространстве в афинной системе координат O_{xyz} уравнение плоскости π, проходящей через точку $M_0(x_0, y_0, z_0)$ с направляющими векторами $\overline{p_i} = \{l_i, m_i, k_i\}$ $(i = 1, 2)$ имеет вид:

$$\begin{vmatrix} x - x_0 & y - y_0 & z - z_0 \\ l_1 & m_1 & k_1 \\ l_2 & m_2 & k_2 \end{vmatrix} = 0$$

Параметрические уравнения.

$\sqsupset$ r = $\overline{OM}$, $r_0 = \overline{OM_0}$ – радиус-векторы точек M и M_0 относительно полюса O. Если $\overline{a}$ - направляющий вектор, то $\overline{M_0M} = t\overline{a},\ t \in \mathbb{R}$. Тогда $\overline{M_0M} = r - r_0$ может быть записано в виде:
Уравнение прямой, проходящей через точку $M_0(\overline{r}_0)$ с направляющим вектором $\overline{a} = \{l, m\}$ имеет вид:

$$\overline{r} = \overline{r}_0 + \overline{a}t,\ t \in \mathbb{R}$$

Уравнение плоскости, проходящей через точку $M_0(\overline{r}_0)$ с направляющими векторами $\overline{p}_i = \{l_i, m_i, k_i\}$, $(i = 1, 2)$ имеет вид:

$$\overline{r} = \overline{r}_0 + u\overline{p}_1 + v\overline{p}_2,\ u, v \in \mathbb{R}$$

Общие уравнения.

Теорема. Линия на плоскости (или поверхность в пространстве) — есть прямая (плоскость) $\iff$ она является алгебраической линией (поверхностью) первого порядка, т. е. задается уравнением

$$Ax + By + C = 0,\ A^2 + B^2 \neq 0$$

$$(Ax + By + Cz + D = 0,\ A^2 + B^2 + C^2 \neq 0 \text{ соответственно})$$

Это уравнение называется **общим уравнением** прямой на плоскости (плоскости в пространстве).
▲ $(\implies)$ l — прямая, каноническое уравнение прямой $\begin{vmatrix} x - x_0 & y - y_0 \\ l & m \end{vmatrix} = 0 \implies mx - ly - mx_0 + ly_0 = 0,\ A = m,\ B = -l,\ C = -mx_0 + ly_0$, при этом $A^2 + B^2 \neq 0$, так как $\overline{a} \neq 0$.
$(\impliedby)$ Рассмотрим линию $l :\ Ax + By + C = 0,\ A^2 + B^2 \neq 0$. Заметим, что при $x_0 = -\dfrac{AC}{A^2 + B^2}$, $y_0 = -\dfrac{BC}{A^2 + B^2}$ точка $M_0(x_0, y_0) \in l$. Вычитая из уравнения линии $Ax + By + C = 0$ уравнение $Ax_0 + By_0 + C = 0$, получим $A(x - x_0) + B(y - y_0) = 0$, а значит $\begin{vmatrix} x - x_0 & y - y_0 \\ -B & A \end{vmatrix} = 0$ — каноническое уравнение прямой, проходящей через точку M_0, с направляющим вектором $\{-B, A\}$, а значит линия l — прямая. ■
Взаимное расположение прямой и плоскости.
$\sqsupset$ плоскость π задана общим уравнением $Ax + By + Cz + D = 0$, а прямая l задана каноническим уравнением $\dfrac{x - x_0}{m} = \dfrac{y - y_0}{n} = \dfrac{z - z_0}{k}$, $M(x_0, y_0, z_0) \in l$, $\overline{p} = (A, B, C)$ – вектор нормали. Тогда:

1. Прямая l принадлежит плоскости $\pi \iff$
   $\begin{cases} M \in \pi \implies Ax_0 + By_0 + Cz_0 + D = 0 \\ (l, \overline{p}) = 0 \implies Am + Bn + Ck = 0 \end{cases}$

2. Прямая l параллельна плоскости $\pi \iff$
   $\begin{cases} M \notin \pi \implies Ax_0 + By_0 + Cz_0 + D \neq 0 \\ (l, \overline{p}) = 0 \implies Am + Bn + Ck = 0 \end{cases}$

3. Прямая l перпендикулярна плоскости $\pi \iff$
   $l \| \overline{p} \implies \dfrac{A}{m} = \dfrac{B}{n} = \dfrac{C}{k}$.

4. Угол φ между прямой l и плоскостью π:

$$sin\varphi = sin(l, \pi) = \frac{|(l, \overline{p})|}{|l| \cdot |\overline{p}|} = \frac{|Am + Bn + Ck|}{\sqrt{A^2 + B^2 + C^2} \cdot \sqrt{m^2 + n^2 + k^2}}$$

Основные задачи на прямую и плоскость.
Углом между двумя прямыми в пространстве называется любой из углов между параллельными им прямыми, проходящими через какую-либо точку пространства. Таким образом, две прямые в пространстве образуют между собой два различных угла в сумме равных π. Угол между направляющими вектоами прямых равен одному из этих углов. Угол между прямыми $l_i : r = r_i + ta_i, i = 1, 2$, совпадающий с угом между направляющими векторами $a_i = \{m_i, n_i, k_i\}$:

$$cos\varphi = \frac{m_1m_2 + n_1n_2 + k_1k_2}{\sqrt{m_1^2 + n_1^2 + k_1^2}\sqrt{m_2^2 + n_2^2 + k_2^2}}$$

Углом между прямой и плоскостью (если они не перпендикулярны) называется меньший из углов между этой прямой и ее ортогональной проекцией на плоскость. Если же прямая и плоскость перпендикулярны, то угол между ними считается равным $\pi/2$. Угол φ между прямой $l : r = r_0 + ta$ и плоскостью $\pi : Ax + By + Cz + D = 0$ находится как дополнительный к углу между направляющим вектором прямой $\overline{a} = \{m, n, k\}$ и вектором нормали к плоскости $\overline{n} = \{A, B, C\}$ и вычисляется:

$$sin\varphi = \frac{|Am + Bn + Ck|}{\sqrt{m^2 + n^2 + k^2}\sqrt{A^2 + B^2 + C^2}},\ 0 \leq \varphi \leq \pi/2$$

Расстояние $\rho(M_1, l)$ от точки $M_1(r_1)$ до прямой $l : r = r_0 + ta$ находится как высота h параллелограмма, построенного на векторах a и $\overline{M_0M_1}$ площадь и основание которого известны

$$\rho(M_1, l) = \frac{|[a, r_1 - r_0]|}{|a|}$$

Расстоянием между скрещивающимися прямыми $l_i : r = r_i + ta_i,\ i = 1, 2$ называется расстояние между параллельными плоскостями, в которых лежат прямые l_1, l_2. Это расстояние $\rho(l_1, l_2)$ находится как высота параллелепипеда, построенного на векторах $\overline{M_1M_2}, a_1, a_2$, объем и площадь основания которого известны:

$$\rho(l_1, l_2) = \frac{|(r_2 - r_1, a_1, a_2)|}{|[a_1, a_2]|}$$

Найти уравнение прямой l: l проходит через т. $M_0(x_0, y_0, z_0)$ и перпендикулярна плоскости $Ax + By + Cz + D = 0$. Ответ: $\dfrac{x - x_0}{A} = \dfrac{y - y_0}{B} = \dfrac{z - z_0}{C}$.
Найти уравнение плоскости π: π проходит через т. $M_0(x_0, y_0, z_0)$ и перпендикулярна прямой $\dfrac{x - x_1}{m} = \dfrac{y - y_1}{n} = \dfrac{z - z_1}{k}$. Ответ: $m(x - x_0) + n(y - y_0) + k(z - z_0) = 0$.
Найти уравнение плоскости π: π проходит через прямую $\dfrac{x - x_1}{m} = \dfrac{y - y_1}{n} = \dfrac{z - z_1}{k}$ и через т. $M_0(x_0, y_0, z_0)$, не лежащую на этой прямой. Ответ:

$$\begin{vmatrix} x - x_0 & y - y_0 & z - z_0 \\ x_1 - x_0 & y_1 - y_0 & z_1 - z_0 \\ m & n & k \end{vmatrix} = 0$$

Найти уравнение плоскости π: π проходит через прямую $\dfrac{x - x_1}{m_1} = \dfrac{y - y_1}{n_1} = \dfrac{z - z_1}{k_1}$ и параллельна другой данной прямой $\dfrac{x - x_2}{m_2} = \dfrac{y - y_2}{n_2} = \dfrac{z - z_2}{k_2}$ (две данные прямые не параллельны). Ответ:

$$\begin{vmatrix} x - x_1 & y - y_1 & z - z_1 \\ m_1 & n_1 & k_1 \\ m_2 & n_2 & k_2 \end{vmatrix} = 0$$

Найти уравнение плоскости π: π проходит через две данные точки $M_0(x_0, y_0, z_0)$ и $M_1(x_1, y_1, z_1)$ и перпендикулярна данной плоскости $Ax + By + Cz + D = 0$ (прямая M_0M_1 и данная плоскость не перпендикулярны). Ответ:

$$\begin{vmatrix} x - x_0 & y - y_0 & z - z_0 \\ x_1 - x_0 & y_1 - y_0 & z_1 - z_0 \\ A & B & C \end{vmatrix} = 0$$

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

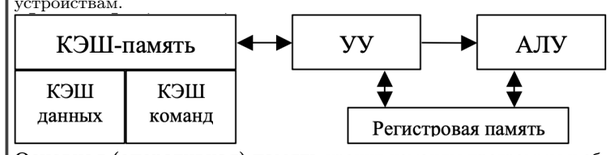
Компьютер — исполнитель алгоритма на языке машины.
Архитектура ЭВМ — совокупность узлов машины и взаимосвязей между ними, рассматриваемая на определённом уровне рассмотрения этой архитектуры.
Принципы фон Неймана:

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 операция и операнды, над которыми должна выполниться данная операция. Выполнение компьютером программы — это автоматическое выполнение определенной последовательности команд, составляющих программу. В компьютере устройство, обеспечивающее выполнение команд, — Последовательность выполняемых процессором команд последовательностью команд и данных, составляющих программу. То есть, по сути, второй принцип – это принцип последовательной обработки.

3. Принцип хранимой 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 команде 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 компиляции программы, когда одна программа порождает другую программу, которая будет выполняться.

Процессор состоит из *устройства управления (УУ)* и *арифметико-логического устройства (АЛУ)*. АЛУ выполняет различные операции над данными, хранящимися на регистрах АЛУ. УУ выполняет команды языка машины, посылая управляющие 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 фиксированный для данной ЭВМ размер. Размер машинного слова - количество двоичных разрядов, размещаемых в машинном слове. Поле машинной информации (тег) - поле ячейки памяти, в котором схемами контроля процессами 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

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 прерывания, 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 прерывания 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.

osn 19. 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 программ (“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 некоторый виртуальной машины.
2. **Интерпретатор** выполняет программы без необходимости предварительной компиляции в машинный код. Может содержать **Just-in-Time** (**JIT**) компилятор, который транслирует программу 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 выполняется непосредственно трансляцией 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, порождаемых компилятором, а также файлов библиотек (которые являются наборами обьектных модулей внутри одного файла), входящих в состав СП.
6. **Отладчик** используется для проверочных запусков программ и исправления ошибок. В нем обычно присутствуют такие возможности как интроспекция (получение типов данных) и анализ данных программы во 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 наиболее количество ресурсов (пример: анализ затрачиваемого на выполнение каждой функции времени, возможно в процентах от полного временем выполнения программы). Используется для оптимизации программ.
6. **Справочная система** содержит справочные материалы по языку программирования и компонентам СП.
7. **Инструменты для статического анализа кода** позволяет найти дефекты в программном коде без выполнения программы с помощью формальных методов.
8. **Средства навигации по коду** позволяют более эффективно ориентироваться в коде и поддерживают, 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 взаимозаменяемы (Например, tiny с compiler может либо интерпретировать Си, либо компилировать), либо, если интерпретатор это виртуальная машина, должны использоваться одновременно (Например, java: javac+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 значащие последовательности – **лексемы**. *Используется разбор с использованием регулярных грамматик для преобразования потока символов в поток лексейм.*

2. **Синтаксический анализ.** Синтаксический анализатор формирует из последовательности лексем (токенов) промежуточное представление программы (синтаксическое дерево, AST, dop24). *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 пометка*

4. **Генерация промежуточного кода.** Перевод AST в некоторое промежуточное представление, на котором удобнее производить оптимизации. *Часто применяется SSA-форма (single static assignment), в которой каждой переменной можно присвоить значение лишь единожды. Для ее построения используется обход графа для генерации трехадресного кода и алгоритм преобразования генерации ϕ-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 графами, см. билеты dop25, dop26.*

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 генерация объектного файла. *Также используются различные алгоритмы работы с графами. 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 использования знаний компилятора для выбора наиболее эффективной инструкции для данной инструкции промежуточного кода.*

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

Общая схема функционирования основных компонентов СП на базе интерпретатора:

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

Основные понятия реляционной модели данных:

• Понятие **типа данных** в реляционной модели данных полностью соответствует понятию типа данных в языках программирования, состоит из трех основных компонентов: определение множества значений данного типа; определение набора операций, применимых к значениям типа; определение способа внешнего представления значений типа (литералов).

• **Домен** — допустимое потенциальное, ограниченное подмножество значений данного типа.

• Для уточнения термина отношение выделяются понятия заголовка отношения, значения отношения и переменной отношения. Пусть дана совокупность типов данных $T_1, T_2, \ldots, T_n$, называемых также доменами, не обязательно различных. Тогда n-арным **отношением** R, или отношением R степени n называют подмножество декартова произведения множеств $T_1, T_2, \ldots, T_n$.

• **Тело отношения** – это множество кортежей вида $\{< A_1, T_1, v_1 >, < A_2, T_2, v_2 >, \ldots, < A_n, T_n, v_n >\}$, где $v_i \in T_i$, A_i – столбцы (атрибуты) отношения.

Алгебра Кодда – основные реляционные операции

• При выполнении операции **объединения** (*UNION*) двух отношений производится отношение, включающее все кортежи, входящие хотя бы в одно из отношений-операндов.

• Операция **пересечения** (*INTERSECT*) двух отношений производит отношение, включающее все кортежи, входящие в оба отношения-операнда.

• Отношение, являющееся **разностью** (*MINUS*) двух отношений, включает все кортежи, входящие в отношение-первый операнд, такие, что ни один из них не входит в отношение, являющееся вторым операндом.

• При выполнении **декартова произведения** (*TIMES*) двух отношений производится отношение, кортежи которого являются конкатенацией (сцеплением) кортежей первого и второго операндов.

• Результатом **ограничения** (*WHERE*) отношения по некоторому условию является отношение, включающее кортежи отношения-операнда, удовлетворяющее этому условию.

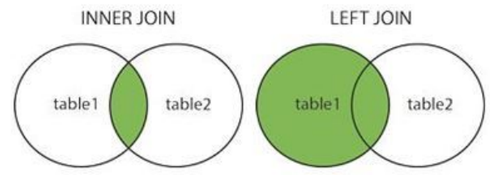
• При выполнении **проекции** (*PROJECT*) отношения на заданное подмножество множества его атрибутов производится отношение, кортежи которого производятся путем взятия соответствующих значений из кортежей отношения-операнда.

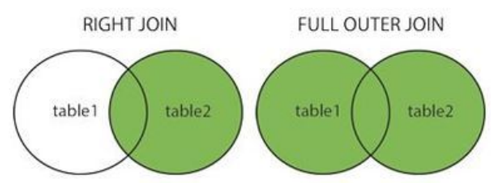
• При **соединении** (*JOIN*) двух отношений по некоторому условию образуется результирующее отношение, кортежи которого являются конкатенацией кортежей первого и второго отношений и удовлетворяют этому условию.

• У операции **реляционного деления** (*DIVIDE BY*) два операнда – бинарное и унарное отношения. Результирующее отношение состоит из унарных кортежей, включающих значения первого атрибута кортежей первого операнда таких, что множество значений второго атрибута (при фиксированном значении первого атрибута) включает множество значений второго операнда.

• Операция **переименования** (*RENAME*) производит отношение, тело которого совпадает с телом операнда, но имена атрибутов могут быть изменены.

• Операция **присваивания** ($:=$) позволяет сохранить результат вычисления реляционного выражения в существующем отношении БД.





Алгебра Кодда – специальные реляционные операции
Имеются важные частные случаи соединения – **эквисоединение** (*EQUIJOIN*) и **естественное соединение** (*NATURAL JOIN*).

• Операция соединения называется операцией **эквисоединения**, если условие соединения имеет вид (a = b), где a и b – атрибуты разных операндов соединения.

• Пусть AB обозначает объединение заголовков отношений A и B. Тогда **естественное соединение** A и B – это спроецированный на AB результат эквисоединения A и B по условию A.c = B.c.

Язык SQL
SQL — универсальный компьютерный язык, применяемый для создания, модификации и управления данными в реляционных базах данных. SQL основывается на реляционной алгебре. Язык SQL представляет собой совокупность операторов, инструкций и вычисляемых функций. Операторы SQL делятся на:

1. **операторы определения данных** (Data Definition Language, DDL):

   – *CREATE* создает объект БД (саму базу, таблицу, представление, пользователя и т. д.);

   – *ALTER* изменяет объект;

   – *DROP* удаляет объект.

2. **операторы манипуляции данными** (Data Manipulation Language, DML):

   – *SELECT* считывает данные, удовлетворяющие условиям;

   – *INSERT* добавляет новые данные;

   – *UPDATE* изменяет существующие данные;

   – *DELETE* удаляет данные.

3. **операторы определения доступа к данным** (Data Control Language, DCL):

   – *GRANT* предоставляет пользователю (группе) разрешения на определенные операции с объектом;

   – *REVOKE* отзывает ранее выданные разрешения;

   – *DENY* задает запрет, имеющий приоритет над разрешением.

4. **операторы управления транзакциями** (Transaction Control Language, TCL):

   – *COMMIT* применяет транзакцию;

   – *ROLLBACK* откатывает все изменения, сделанные в контексте текущей транзакции;

   – *SAVEPOINT* делит транзакцию на более мелкие участки.

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';
```

[Кузнецов, *Основы современных БД*]

osn 23. Основные методы обработки изображений: тональная коррекция, свёртка изображений, выделение краёв.

• **Тональная коррекция** заключается в перераспределении света и тени между пикселями, то есть в регулировке яркости и контрастности изображения.

Как мы можем численно оценить, контрастным получилось изображение или нет? При помощи оценки **гистограммы яркости** – это график, где по горизонтальной оси отложены уровни яркости от 0 до 255, а по вертикальной оси число пикселей изображения.

Для определения яркости пикселя можно воспользоваться формулой Y=0.3*R+0.59*G+0.11*B, правда это больше для изображений в оттенках серого.

• Часть диапазона, которая используется в изображении, называется его **тоновым диапазоном**.

Крайняя слева точка гистограммы соответствует нулевой, а крайняя справа – максимально возможной яркости. Чем шире тоновый диапазон, тем больше деталей может передать изображение и тем выше его качество. Условно тоновый диапазон делится на три части. Левая часть соответствует темным местам изображения, правая часть — светлым местам, между ними находятся средние тона. В идеальном изображении присутствуют пиксели всех оттенков.

• Что можно выявить:

– Не полностью используется диапазон яркости;

– Концентрация яркости вокруг определенных значений — неравномерное заполнение диапазона яркости.

Методы решения проблемы

• **Линейная коррекция.** $f^{-1}(x) = y$; x — яркость пикселя на исходном изображении, y — яркость пикселя после коррекции. Компенсация узкого диапазона яркостей — линейное растяжение гистограммы (histogram equalization):

$$y = \frac{(x - x_{min}) \cdot (255 - 0)}{(x_{max} - x_{min})}$$

• **Нелинейная коррекция**:
Гамма-коррекция. Изначальная цель – коррекция для правильного отображения на мониторе, $y = c \cdot x^{\gamma}$;

Логарифмическая. Цель – сжатие динамического диапазона при визуализации данных, $y = c \cdot \log(1+x)$. Тут y — яркость уже после коррекции.

Свертка — это операция вычисления нового значения выбранного пикселя, учитывающая значения окружающих его пикселей.
Для вычисления значения используется матрица, называемая ядром свертки. Обычно ядро свертки является квадратной матрицей n * n, где n — нечетное, однако ничто не мешает сделать матрицу прямоугольной. Во время вычисления нового значения выбранного пикселя ядро свертки как бы <прикладывается> своим центром (именно тут важна нечетность размера матрицы) к данному пикселю. Окружающие пиксели так же накрываются ядром. Далее высчитывается сумма, где слагаемыми являются произведения значений пикселей на значения ячейки ядра, накрывшей данный пиксель. Сумма делится на сумму всех элементов ядра свертки. Полученное значение как раз и является новым значением выбранного пикселя. Если применить свертку к каждому пикселю изображения, то в результате получится некий эффект, зависящий от выбранного ядра свертки.
Пространственная фильтрация. Линейные фильтры
Под **фильтрацией** изображений понимают операцию, имеющую своим результатом изображение того же размера, полученное из исходного изображения по некоторым правилам.
Обычно интенсивность (цвет) каждого пикселя результирующего изображения обусловлена интенсивностями (цветами) пикселей, расположенных в некоторой его окрестности в исходном изображении. Это надо для борьбы с шумом на изображении.
Алгоритм: Заменим каждый пиксель взвешенным средним по окрестности. Веса обозначаются как ядро фильтра. Веса для усреднения задаются в виде квадратной матрицы k * k.
Пусть f — изображение, g — ядро. Свертка f по g тогда обозначается как $f * g$: $(f * g)[m, n] = \sum f(m-k, n-l) * g(k, l)$, где k, l пробегают число строк/столбцов матрицы g — так и получаем усреднение.
Основные свойства линейных фильтров:

• Линейность:

$filter(f_1 + f_2) = filter(f_1) + filter(f_2)$

$filter(a * f_1) = a * filter(f_1)$

• Инвариантность к сдвигу — не зависит от сдвига пиксела:

$filter(shift(f)) = shift(filter(f))$

Теорема. Любой линейный оператор, инвариантный к сдвигу, может быть записан в виде свертки.

Примеры фильтров:

• box-фильтр;

• Фильтр Гаусса;

• Медианный — меняет значение на медиану по фильтру (это нелинейный фильтр — вдруг спросят);

• unsharp (повышение резкости)

Линейная фильтрация (свёртка) изображения позволяет решать целый ряд задач — шумоподавление, повышение резкости, оценка градиента.

Выделение краев
Задача: выделить резкие изменения (разрывы) изображения. Край — это точка резкого изменения значений функции интенсивности изображения.

• Градиент изображения: $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$. Градиент направлен в сторону наибольшего изменения интенсивности.

• Сила края задается величиной (нормой) градиента:

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

• Разностная производная (для функции 2-х переменных):
$\frac{\partial f}{\partial x} \approx \frac{f(x_{n+1}, y) - f(x_n, y)}{\Delta x}$ – это свёртка.

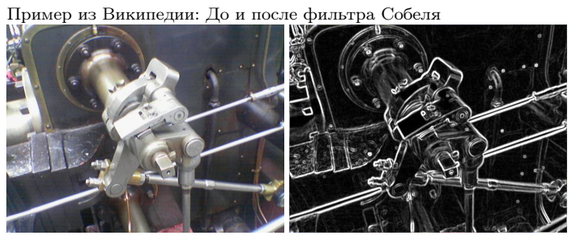
Задачу выделения краёв изображения в простом случае можно решить как поиск локальных максимумов градиента яркости.
После получения силы края во всех точках можно по-разному определять края – отсекать по пороговому значению, выделять локальные максимумы для утоньшения краёв и повышения точности, отсекать по гистерезису (для начала построения кривой границы порог большой, для продолжения порог ниже).
Фильтр Собеля (Sobel filter)
Используются свертки 3*3, с помощью которых аппрокимируются производные – горизонтальная и вертикальная. Пусть F – исходное изображение, G_x and G_y – изображения которые в каждой точке содержат приближение горизонтальной и вертикальной производных.

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * F, \; G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * F$$

где * означает операцию свертки.
x здесь «увеличивается» вправо, y - вниз. В каждой точке можно объединить их и получить модуль градиента.

$$G = \sqrt{G_x^2 + G_y^2}$$

Пример из Википедии: До и после фильтра Собеля



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

osn 24. Линейные обыкновенные дифференциальные уравнения и системы. Фундаментальная система решений. Определитель Вронского.

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

$$\mathcal{L}y = f(t) = a_0(t)y^{(n)}(t) + a_1(t)y^{(n-1)}(t) + \cdots + a_{n-1}(t)y'(t) + a_n(t)y(t),$$

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 $\mathcal{L}y = f(t),\ y^{(k)}(t_0) = y_{0k},\ k = \overline{0, n-1}$ 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 $v^{(k)}(t_0) = 0, k = \overline{0, n-1}$, а функция $w(t)$ является решением задачи Коши для *однородного* уравнения $\mathcal{L}w = 0$ с ненулевыми начальными условиями $w^{(k)}(t_0) = y_{0k},\ k = \overline{0, n-1}$.

▲ Сумма $y(t) = v(t) + w(t)$ удовлетворяет неоднородному ур-ю в ситу 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 $\varphi_1(t), \ldots, \varphi_m(t)$ называются **линейно зависимыми** на отрезке $[a,b]$, если найдутся такие комплексные константы $c_k \in \mathbb{C},\ k = \overline{1,m}, \sum_{k=1}^{m} |c_k| > 0$, что справедливо равенство $\sum_{k=0}^{m} c_k \varphi_k(t) = 0,\ \forall t \in [a,b]$. Если равенство выполнено только для $c_k = 0,\ k = 1..n$, то функции **линейно независимы**.

Определителем Вронского (вронскианом) системы функций $\varphi_1(t), \ldots, \varphi_m(t)$, где $\varphi_i(t) \in C^{(m-1)}[a,b]$, называется зависящий от переменной $t \in [a,b]$ определитель

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

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

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

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

Отсюда следует, что вектор-столбцы определителя Вронского 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 $W \neq 0$, контрпример: $\varphi_1(t) = t^2,\ \varphi_2(t) = \{-t^2, t < 0; t^2, t \geq 0\}, W \equiv 0$, но на $[-1,1]$ функции ЛНЗ.

Теорема 4: Для решений $y_1(t), \ldots, y_n(t)$ линейного однородного уравнения на отрезке $[a,b]$ справедлива следующая **альтернатива**:
либо $W[y_1, \ldots, y_n](t) = 0$ на $[a,b]$ и функции $y_1(t), \ldots, y_n(t)$ линейно зависимы на этом отрезке;
либо $W[y_1, \ldots, y_n](t) \neq 0, \forall t \in [a,b]$ и функции $y_1(t), \ldots, y_n(t)$ 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 решение 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 $b_{ij}, i,j = 1,2,\ldots,n$ такую, что $detB \neq 0$. Обозначим через $y_j(t)$ решения задачи Коши для уравнения $\mathcal{L}y = 0$ с начальными условиями:

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

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 Вронского $W[y_1, \ldots, y_n](t)$, в силу начальных условий, таков, что $W[y_1, \ldots, y_n](t_0) = detB \neq 0$. 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 $y_1(t), y_2(t), \ldots, y_n(t)$ — 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:

$$y_{OO}(t) = c_1y_1(t) + c_2y_2(t) + \cdots + c_ny_n(t),\ \forall c_j \in \mathbb{C}. \qquad (5)$$

▲ Так как линейная комбинация решений однородного уравнения $\mathcal{L}y = 0$ является решением этого уравнения, то при любых значениях постоянных c_k функция $y_{OO}(t)$, определяемая формулой (5), является решением линейного однородного дифференциального уравнения $\mathcal{L}y = 0$.
Покажем теперь, что любое решение уравнения $\mathcal{L}y = 0$ может быть получено из (5) 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:

$$\begin{cases} c_1y_1(t_0) + c_2y_2(t_0) + \cdots + c_ny_n(t_0) = \tilde{y}(t_0) \\ c_1y_1'(t_0) + c_2y_2'(t_0) + \cdots + c_ny_n'(t_0) = \tilde{y}'(t_0) \\ \ldots \\ c_1y_1^{(n-1)}(t_0) + c_2y_2^{(n-1)}(t_0) + \cdots + c_ny_n^{(n-1)}(t_0) = \tilde{y}^{(n-1)}(t_0) \end{cases} \qquad (6)$$

где t_0 — некоторая точка отрезка $[a,b]$. Определитель этой системы равен определителю Вронского в точке t_0 и не равен 0, так как решения $y_1(t), y_2(t), \ldots, y_n(t)$ линейно независимы. Следовательно, система (6) имеет единственное решение $\tilde{c}_1, \tilde{c}_2, \ldots, \tilde{c}_n$.

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 $\tilde{c}_1, \tilde{c}_2, \ldots, \tilde{c}_n$ представляют собой решение системы (6), то функция $\hat{y}(t)$ такова, что $\hat{y}^{(k)}(t_0) = \tilde{y}^{(k)}(t_0),\ k = 0,1,\ldots,n-1$.
Следовательно, функции $\hat{y}(t)$ и $\tilde{y}(t)$ являются решениями уравнения $\mathcal{L}y = 0$ и удовлетворяют одним и тем же начальным условиям в точке $t0$. По теореме о существовании и единственности решения задачи Коши эти функции должны совпадать: $\hat{y}(t) = \tilde{y}(t) = \sum_{k=1}^{n} \tilde{c}_k y_k(t)$ ■

Теорема 7: Пусть $y_1(t), y_2(t), \ldots, y_n(t)$ — 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:

$$y_{OH}(t) = y_H(t) + y_{OO}(t) = y_H(t) + c_1y_1(t) + c_2y_2(t) + \cdots + c_ny_n(t), \qquad (7)$$

где $c_1, c_2, \ldots, c_n$ — произвольные комплексные постоянные.

▲ Для любого набора констант $c_j \in \mathbb{C}$ формула (7) определяет решение линейного неоднородного уравнения $\mathcal{L}y = f(t)$ в силу линейности уравнения. Согласно определению общего решения осталось показать, что выбором констант в (7) 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 $\tilde{c}_1, \tilde{c}_2, \ldots, \tilde{c}_n$ такие, что на отрезке $[a,b]$ будет выполнено равенство

$$\tilde{y}(t) = y_H(t) + \tilde{c}_1y_1(t) + \tilde{c}_2y_2(t) + \cdots + \tilde{c}_ny_n(t). \qquad (8)$$

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 $\tilde{y}(t) = \tilde{c}_1y_1(t) + \tilde{c}_2y_2(t) + \cdots + \tilde{c}_ny_n(t)$, а вместе с ним и искомое равенство (8). ■

[А. М. Денисов, *Обыкновенные дифференциальные уравнения, часть 1*, page 65-73]

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 тысячу 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, выполнив свою работу, передавал бы результат следующему, одновременно принимая новую порцию входных данных. Получаем выигрыш в скорости обработки за счет совмещения прежде разнесенных во времени операций. Существует некоторая задержка (время разгона), для того, чтобы заполнить все 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** (циркулирующий между оперативной памятью и внешними устройствами). В связи с этим выделяют 4 основных класса: SISD (1 поток команд, 1 поток данных. "Традиционный" последовательный компьютер), SIMD (1 поток команд, много потоков данных, пример - векторные компьютеры), MISD (много п. ком., 1 п. данных), MIMD (много и потоков команд, 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 единым адресным пространством. (+) относительная простота параллельного программирования; (–) сложность увеличения числа процессоров (роста производительности).

UMA – системы с однородным доступом к памяти (все процессоры имеют одинаковый доступ к памяти). SMP — есть общая шина, соединенная со всеми процессорами и с ОП.

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

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

• **Компьютеры с распределенной памятью.**

Состоят из вычислительных узлов, каждый из которых является полноценным компьютером со своей памятью, ОС, устройствами вводавывода и т.п., взаимодействующих друг с другом через коммуникационную среду. (–) сложность параллельного программирования; (+) относительная простота увеличения числа процессоров (роста производительности).

Основные понятия для распределённых систем

• **Длина критического пути** – минимальное количество элементарных связей, которые нужно пройти для коммутации двух самых удаленных процессоров. что такое коммутация процессов?

• **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 по каналам связи и измеряется объёмом передаваемой информации в единицу времени. Время на передачу сообщения по коммуникационной сети вычисляется по следующей формуле: $t_N = t_0 + \frac{N}{S}$, где t_0 – латентность, N – объём передаваемых данных, S – пропускная способность сети.

Схема коммутации	Длина критического пути	Связность	Сложность
линейка	$p-1$	1	$p-1$
кольцо	$\lfloor \frac{p}{2} \rfloor$	2	p
звезда	2	1	$p-1$
полносвязная топология	1	$p-1$	$\frac{p(p-1)}{2}$

p – количество процессов

Методы оценки производительности.

• **Пиковая производительность** – теоретический предел производительности для данного компьютера. Пиковая производительность компьютера вычисляется как сумма пиковых производительностей всех входящих в него вычислительных устройств (процессоров, ускорителей и т.д.). Даёт нижнюю оценку времени выполнения программы. Производительность компьютера на любой реальной программе никогда не только не превысит этого порога, но и не достигнет его точно.

• **Реальная производительность** – производительность данного компьютера на конкретном приложении. Традиционно используются два способа оценки производительности компьютера. Один из них опирается на число команд, выполняемых компьютером в единицу времени. Единица измерения – **MIPS** (Million Instructions Per Second). Второй способ – число вещественных операций, выполняемых компьютером в единицу времени – **Flops** (Floating point operations per second).

Популярные тесты:

– **LINPACK**: измерение производительности при обработке чисел с плавающей точкой. Задача: решение СЛАУ.

– **Graph500**: нагружает коммуникационную систему компьютера и не зависит от количества исполняемых в секунду операций с числами с плавающей точкой. Задача: поиск в ширину в большом ненаправленном графе.

– **NAS Parallel Benchmark**: набор различных задач для проверки производительности.

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

Основные механизмы (постулаты) ООП:

1. **Инкапсуляция** – это *механизм*, который связывает данные с обрабатывающими их кодами и защищает и те, и другие от внешних воздействий и ошибочных действий. В объектно-ориентированном языке коды и данные могут быть связаны так, что вместе они создают автономный *черный ящик*. Внутри этого ящика содержатся все необходимые данные и коды. При связывании таким образом данных и кодов создается *обвект*. Другими словами, объект представляет собой устройство, поддерживающее инкапсуляцию.
Внутри объекта коды или данные или и те, и другие могут иметь атрибут *private*, что делает их закрытыми для внешнего мира, или *public*, что открывает эти элементы объекта. Закрытые коды и данные известны и доступны только из других частей того же объекта. Другими словами, к закрытым кодам и данным нельзя обратиться ни из какого фрагмента программы, существующего вне объекта. Если же код или данные объявлены с атрибутом public, то доступ к ним открыт из любых частей программы, несмотря на то, что эти элементы определены внутри объекта. Обычно открытые элементы объекта используются для обеспечения контролируемого интерфейса с закрытыми элементами того же объекта.
В C++ базовой единицей инкапсуляции является **класс**. Класс определяет содержание объекта. Класс описывает как данные, так и коды, предназначенные для операций над этими данными. C++ использует спецификацию класса при конструировании *обвектов*. Объекты являются экземплярами класса. Т.е. класс в сущности представляет собой набор чертежей, по которым строится объект.
Код и данные, составляющие класс, называются *членами* класса. Конкретно, *члены-переменные*, называемые также переменными экземпляра, – это данные, определенные в классе. *Члены-функции*, или просто функции – это коды, предназначенные для операций над данными.
2. **Полиморфизм** обозначает средство, позволяющее посредством единого интерфейса получить доступ к целому классу действий. Простым примером полиморфизма может служить рулевое колесо автомобиля. Рулевое колесо (интерфейс) остается одним и тем же, независимо от того, какой тип рулевого механизма используется в данном автомобиле. Другими словами, рулевое колесо действует одинаково для любых автомобилей: с непосредственным приводом на колеса, с гидравлическим усилителем или с реечной передачей. Поворот рулевого колеса влево заставляет автомобиль двигаться влево независимо от типа рулевого механизма. Достоинство единого интерфейса, очевидно, заключается в том, что если вы умеете пользоваться рулевым колесом, вы можете ездить на любых автомобилях.
Рассмотрим стек (список, действующий по правилу "первым вошел, последним вышел"). Пусть вашей программе требуется три стека различных видов. Один стек используется для целых чисел, другой для чисел с плавающей точкой, а третий для одиночных символов. Алгоритм реализации всех трех стеков будет одинаков, несмотря на то, что данные, заносимые в разные стеки, различаются.
В общем случае концепция полиморфизма часто выражается фразой "один интерфейс, много методов". Это означает возможность разработать обобщенный интерфейс для группы схожих действий.
Различают **статический** (реализуется на этапе компиляции с помощью перегрузки функций и операций), **динамический** (реализуется во время выполнения программы с помощью механизма виртуальных функций) и **параметрический** (реализуется на этапе компиляции с использованием механизма шаблонов) полиморфизм.
3. **Наследование** является процессом, который позволяет одному объекту приобретать свойства другого объекта. Важность наследования определяется тем, что оно поддерживает концепцию иерархической классификации. Большая часть наших знаний построена по иерархическому принципу. Например, антоновка является частью класса яблок, который, в свою очередь, есть часть класса фруктов; фрукты же входят в более широкий класс пищевых продуктов. Класс пищевые продукты обладает определенными качествами (съедобность, пищевая ценность и т. д.), которые, логично предположить, приложимы и к его подклассу фрукты. В дополнение к этим качествам класс фрукты обладает специфическими характеристиками (сочность, сладость и др.), которые выделяют его среди других пищевых продуктов. Класс яблоки определяет качества, характерные для яблок (растут на деревьях, не являются тропическими продуктами и т. д.). Класс антоновка, в свою очередь, наследует все качества всех предшествующих классов и определяет лишь те качества, которые выделяют антоновку среди прочих яблок.
Если не пользоваться иерархией, то каждый объект должен был бы явно определять все свои характеристики. При использовании же наследования объект определяет лишь те качества, которые делают его уникальным в рамках его класса. Более общие качества он может наследовать от родительского класса. Таким образом, механизм наследования позволяет объекту быть специфическим экземпляром более общего класса.

Примеры на C++:

1. **Инкапсуляция**. Пример класса *«коробка»*, инкапсулирующего данные и предоставляющего метод для вычисления объема.

```
class Box {
    int length;
    int width;
    int height;
public:
    int volume() const {
        return length * width * height;
    }
}
```

2. **Наследование**. Наследование свойств и поведения могут контролироваться с помощью квалификаторов доступа, задаваемых при наследовании: *public, protected, private*. В примере ниже класс C является наследником класса A.

```
class A {
public:
    virtual void f(int x) {
        cout << "A::f" << '\n';
    }
};

class C: public A {
public:
    void f(int x) {
        cout << "C::f" << '\n';
    }
};
```

3. **Полиморфизм.**
Статический полиморфизм реализуется с помощью перегрузки функций и операций. Под перегрузкой функций в C++ понимается описание в одной области видимости нескольких функций с одним и тем же именем.

```
void f(int x);
void f(double x);
```

Динамический полиморфизм реализуется с помощью механизма виртуальных методов. Механизм виртуальных методов заключается в том, что результат вызова виртуального метода с использованием указателя или ссылки зависит не от того, на основе какого типа создан указатель, а от типа объекта, на который он указывает. Тип данных (класс), содержащий хотя бы одну виртуальную функцию, называется **полиморфным типом (классом)**, а объект этого типа – **полиморфным объектом**. Во всех наследниках виртуальная функция остается таковой.
В примере ниже используются описанные выше классы A и C.

```
int main() {
    A a1;
    C c1;
    C *pc = &c1;
    pc->f(1); // C::f
    A *pa = pc;
    pa->f(1); // C::f
    pc = (C*) &a1;
    pc->f(1); // A::f
    return 0;
}
```

Чистая виртуальная функция — функция вида:
virtual <тип_возвращаемого_значения> имя_функции (формальные_параметры) = 0;
Такая форма записи функции означает, что данная функция (точнее, метод класса) не имеет тела, описывающего ее алгоритм.
Абстрактный класс — это класс, содержащий хотя бы одну чистую виртуальную функцию.
Параметрический полиморфизм позволяет применить один и тот же алгоритм к разным типам данных. При этом тип является параметром тела алгоритма. При обращении к функции-шаблону после имени функции в угловых скобках указываются фактические параметры шаблона – имена реальных типов или значения объектов.

```
template <class T> T max(T &x, T &y) {
    return x > y ? x : y;
}
```

[Шилдт, *C++. Шаг за шагом*, page 20-23]

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).

Операционная система — комплекс программ, используемых для управления ресурсами компьютера и предоставления интерфейса пользователю. В понятие управление ресурсами входит *выделение* ресурсов для программ (например, памяти и процессорного времени), *защита* от доступа программ к ресурсам, которыми они не владеют, а также *абстрагирование* 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).
Физические ресурсы (устройства) – компоненты аппаратуры компьютера, используемые на программных уровнях ВС или оказывающие влияние на функционирование всей ВС. Совокупность физических ресурсов составляет аппаратный уровень вычислительной системы.
Логические, или виртуальные ресурсы (устройства) ВС – устройство/ресурс, некоторые эксплуатационные характеристики которого (возможно все) реализованы программным образом.
Состав ОС:

• ядро ОС - резидентная часть ОС, реализующая некоторую базовую функциональность ОС и работающая в режиме супервизора;

• динамически подгружаемые драйверы физических и виртуальных устройств. под динамически подгружаемыми понимается то, что в зависимости от ситуации состав этих драйверов при инсталяции и загрузке системы может меняться;

• интерфейсы системных вызовов.

Типы ОС:
- Пакетные ос - система, критерием эффективности которой является максимальная загрузка ЦП. Время работы процессора/время работы исполнения пользовательских программ 1. Пакет программ - совокупность программ которые системе необходимо обработать. Переключение процессов происходит по 1 из трех причин: зацикливание процесса, завершение процесса, обращение к внешнему устройству.
- ОС разделения времени - модель, представляющая собой развитие пакетных систем. Дополнительная характеристика - квант процессорного времени - некоторый фиксированный ос промежуток времени работы процессора. + причина по смене исполняемого процесса - исчерпание кванта времени.
- ОС реального времени - системы, ориентированные на обработку некоторого фиксированного набора событий, при возникновении любого из которых гарантируется обработка этого события за некоторый промежуток времени, не превосходящий определенного предельного значения.
- Сетевые ОС - ос, обеспечивающая функционирование и взаимодействие вычислительной системы в пределах сети.
- Распределенная система - система, функционирующая на многопроцессорном/многомашинном комплексе, в котором на каждом из узлов функционирует отдельное ядро, а сама система обеспечивает реализацию распределенных возможностей ОС.
Под **процессом** понимается совокупность машинных команд и данных, обрабатываемая в вычислительной системе и обладающая правами на владение некоторым набором ресурсов ВС. **Процесс (полновесный)** - объект планирования и выполняется внутри защищённой области памяти. **Легковесные процессы** - могут активироваться внутри полновесного процесса, могут быть объектами планирования, и при этом они могут функционировать внутри общей (т.е. незащищённой от других нитей) области памяти. Понятие процесса включает в себя следующее:исполняемый код, собственное адресное пространство, представляющее собой множество виртуальных адресов, которые может использовать процесс, ресурсы системы, которые назначены процессу ос, хотя бы одну выполняемую нить. **Системный вызов** - средство ос, предоставляемое пользователям (процессам), посредством которого процессы могут обращаться к ядру ос за выполнением тех или иных функций. **Процесс** - объект, порожденный системным вызовом

fork.
Будем говорить, что процессы называются **параллельными**, если их выполнение хотя бы частично перекрывается по времени. Совместное использование ресурса ВС двумя и более параллельными процессами, когда каждый некоторое время владеет этим ресурсом, называется **разделением ресурса.** Ситуация, когда процессы конкурируют за разделяемый ресурс, называются **гонкой процессов**. **Критический ресурс** - разделяемый ресурс, который в каждый момент времени доступен только одному из взаимодействующих процессов.
Пример. Имеется разделяемый ресурс *in* и два процесса. В некоторый момент времени процесс A присвоил переменной *in* значение X. Затем в некоторый момент процесс B присвоил значение Y этой же переменной *in*. Далее оба процесса читают эту переменную, и в обоих случаях процессы прочтут значение Y . То есть символ, считанный процессом A, был потерян, а символ, считанный процессом B, был выведен дважды. Результат выполнения процессов здесь зависит от того, в какой момент осуществляется переключение процессов, и от того, какой конкретно процесс будет выбран следующим для выполнения.
Взаимное исключение - способ работы с разделяемым ресурсом, при котором когда один из процессов работает с разделяемым ресурсом, все остальные не могут иметь к нему доступ.
Блокировка - доступ к разделяемому ресурсу одного из взаимодействующих процессов не обеспечивается из-за активности более приоритетных.
Тупик - взаимоблокировка.
Семафоры Дейкстры
Имеется специальный тип данных — *семафор*. Переменные типа семафор могут принимать целочисленные значения. Определены атомарные операции: опустить семафор *down(S)* (или *P(S)*) и поднять семафор *up(S)* (или *V(S)*).
Операция *down(S)* проверяет значение семафора S и, если оно больше нуля, то уменьшает его на 1. Если же это не так, процесс блокируется, причем связанная с заблокированным процессом операция down считается незавершенной.
Операция *up(S)* увеличивает значение семафора на 1. При этом если в системе присутствуют процессы, блокированные ранее при выполнении *down* на этом семафоре, то один из них разблокируется и завершает выполнение операции *down*, т.е. вновь уменьшает значение семафора. Выбор процесса для разблокирования никак не оговаривается. Используется для предотвращения тупика.
Монитор - это языковая конструкция с централизованным управлением (в отличие от семафоров, которые не обладают централизацией).

1. Структуры данных монитора доступны только через обращения к процедурам или функциям этого монитора (т.е. монитор представляет собой некоторый аналог объекта в объектно-ориентированных языках и реализует инкапсуляцию данных);
2. процесс занимает (или входит в) монитор, если он вызывает одну из процедур или функций монитора;
3. В каждый момент времени внутри монитора может находиться не более одного процесса.

Механизм передачи сообщений основан на двух функциональных примитивах: *send* и *receive*. Их можно разделить по трем характеристикам: модель синхронизации (операции посылки/приема сообщений могут быть блокирующими и неблокирующими), адресация (прямая (конкретный адрес) или косвенная (сообщение бросается в общий пул)) и формат сообщения.
Сигналы — средство оказания воздействия одним процессом на другой процесс (одним из них может быть ОС). Используются непосредственные имена процессов. Асинхронное взаимодействие (момент прихода сигнала заранее неизвестен). Действия при получении: обработка по умолчанию(процесс завершается с кодом сигнала), специальная обработка(вызывается спец ф-я), игнорирование. Порядок реагирования не определен. Чтобы установить реакцию процесса на приходящий сигнал, используется системный вызов *signal()*.
Неименованый канал (англ. pipe) - это объект, позволяющий реализовать односторонний канал между двумя процессами. Создается вызовом *pipe()*, который возвращает два файловых дескриптора, один на чтение, другой на запись. Один процесс пишет в файловый дескриптор на запись, другой читает из файлового дескриптора на чтение. При этом реального файла в файловой системе нет.
Предельный размер канала декларируется параметрами настройки ОС. Для создания – системный вызов *pipe*(). К неименованному каналу невозможен доступ по имени, существует в системе, пока существуют процессы, его использующие. Предназначен для синхронизации и организации взаимодействия родственных процессов.
IPC (Inter-Process Communication) предоставляет взаимодействующим процессам общие (разделяемые) ресурсы. Например, SytemV IPC предоставляет следующие типы разделяемых ресурсов:

• **Очередь сообщений** - это разделяемый ресурс, позволяющий организовывать очереди сообщений: один процесс может в эту очередь положить сообщение, а другой процесс - прочитать его.

• **Массив семафоров** - ресурс, представляющий собой массив из N элементов, где N задается при создании данного ресурса, и каждый из элементов является семафором IPC.

• **Общая (разделяемая) память** представляется процессу как указатель на область памяти, которая является общей для двух и более процессов.

osn 25. Теоремы существования и единственности решения задачи Коши для обыкновенного дифференциального уравнения первого порядка, разрешенного относительно производной.

Пусть функция $f(t,y)$ определена и непрерывна в прямоугольнике $\Pi = \{(t,y) : |t-t_0| \leqslant T, |y-y_0| \leqslant A\}$.
Рассмотрим на отрезке $[t_0-T; t_0+T]$ дифференциальное уравнение с условием:

$$y'(t) = f(t, y(t)) \tag{9}$$

$$y(t_0) = y_0 \tag{10}$$

Требуется определить функцию $y(t)$, удовлетворяющую уравнению (9) и условию (10).
Эта задача называется **задачей с начальным условием или задачей Коши**. Рассмотрим отрезок $[t_1,t_2]$ такой, что $t_0 - T \leqslant t_1 < t_2 \leqslant t_0+T,\ t_0 \in [t_1,t_2]$.
Опр. Функция $y(t)$ называется **решением задачи Коши** (9), (10) на отрезке $[t_1,t_2]$, если $y(t) \in C^1[t_1,t_2],\ |y(t)-y_0| \leqslant A$ для $t \in [t_1,t_2],\ y(t)$ удовлетворяет уравнению (9) для $t \in [t_1,t_2]$ и (10).
Рассмотрим на отрезке $[t_0-T, t_0+T]$ уравнение относительно неизвестной функции $y(t)$:

$$y(t) = y_0 + \int_{t_0}^{t} f(\tau, y(\tau))d\tau \tag{11}$$

Лемма 1. Функция $y(t)$ является решением задачи Коши (9), (10) на отрезке $[t_1,t_2] \iff$ когда $y(t) \in C[t_1,t_2],\ |y(t)-y_0| \leqslant A$ для $t \in [t_1,t_2]$ и $y(t)$ удовлетворяет уравнению (11) для $t \in [t_1,t_2]$.
▲ ($\Longrightarrow$) Пусть функция $\overline{y}(t)$ является решением задачи с начальным условием (9), (10) на отрезке $[t_1,t_2]$. Из определения решения следует, что $\overline{y}(t) \in C[t_1,t_2],\ |\overline{y}(t)-y_0| \leqslant A$ для $t \in [t_1,t_2]$. Покажем, что $\overline{y}(t)$ удовлетворяет уравнению (11) для $t \in [t_1,t_2]$. Интегрируя (9) от t_0 до t получим:

$$\int_{t_0}^{t} \overline{y}'(\tau)d\tau = \int_{t_0}^{t} f(\tau, \overline{y}(\tau))d\tau,\ t \in [t_1,t_2]$$

Учитывая условие (10), имеем:

$$\overline{y}(t) = y_0 + \int_{t_0}^{t} f(\tau, \overline{y}(\tau))d\tau,\ t \in [t_1,t_2]$$

Следовательно, функция $\overline{y}(t)$ удовлетворяет интегральному уравнению (11) при $t \in [t_1,t_2]$.
($\Longleftarrow$) Пусть функция $\overline{y}(t)$ такова, что $\overline{y}(t) \in C[t_1,t_2],\ |y(t)-y_0| \leqslant A$ для $t \in [t_1,t_2]$ и $\overline{y}(t)$ удовлетворяет уравнению (11) для $t \in [t_1,t_2]$, то есть:

$$\overline{y}(t) = y_0 + \int_{t_0}^{t} f(\tau, \overline{y}(\tau))d\tau,\ t \in [t_1,t_2] \tag{12}$$

Покажем, что $y(t)$ является решением задачи с начальным условием (9), (10).
Положив в (12) $t = t_0$, получим, что $\overline{y}(0) = y_0$. Следовательно условие (10) выполнено. Так как функция $\overline{y}(t)$ непрерывна на $[t_1,t_2]$, то правая часть равенства (12) непрерывно дифференцируема на $[t_1,t_2]$ как интеграл с переменным верхним пределом t от непрерывной функции $f(\tau, \overline{y}(\tau)) \in C[t_1,t_2]$. Следовательно, $\overline{y}(t)$ непрерывно дифференцируема на $[t_1,t_2]$. Дифференцируя (12), получим, что $\overline{y}(t)$ удовлетворяет (9). ■
Опр. Функция $f(t,y)$, заданная в прямоугольнике Π, *удовлетворяет в Π условию Липшица* по y, если $|f(t,y_1) - f(t,y_2)| \leqslant L|y_1 - y_2|,\ \forall (t,y_1),(t,y_2) \in \Pi$, где L — положительная постоянная.
Лемма Гронуолла-Беллмана. Пусть функция $z(t) \in C[a,b]$ и такова, что $0 \leqslant z(t) \leqslant c + d\left|\int_{t_0}^{t} z(\tau)d\tau\right|,\ t \in [a,b]$, где постоянная c неотрицательна, постоянная d положительна, а t_0 — произвольное фиксированное число на отрезке $[a,b]$. Тогда $z(t) \leqslant ce^{d|t-t_0|},\ t \in [a,b]$.
Теорема (единственности). Пусть функция $f(t,y)$ непрерывна в Π и удовлетворяет в Π условию Липшица по y. Если $y_1(t),\ y_2(t)$ — решения задачи Коши (9), (10) на отрезке $[t_1,t_2]$, то $y_1(t) = y_2(t)$ для $t \in [t_1,t_2]$.
▲ Так как $y_1(t)$ и $y_2(t)$ — решения задачи Коши (9), (10), то из Леммы 1 следует, что они являются решениями интегрального уравнения (11). То есть:

$$y_1(t) = y_0 + \int_{t_0}^{t} f(\tau, y_1(\tau))d\tau,\ t \in [t_1,t_2],$$

$$y_2(t) = y_0 + \int_{t_0}^{t} f(\tau, y_2(\tau))d\tau,\ t \in [t_1,t_2].$$

Вычитая второе уравнение из первого и оценивая разность по модулю и используя условие Липшица, получаем: $|y_1(t) - y_2(t)| = \left|\int_{t_0}^{t} f(\tau, y_1(\tau))d\tau - \int_{t_0}^{t} f(\tau, y_2(\tau))d\tau\right| \leq \left|\int_{t_0}^{t} |f(\tau, y_1(\tau)) - f(\tau, y_2(\tau))|d\tau\right| \leq L\left|\int_{t_0}^{t} |y_1(\tau) - y_2(\tau)|d\tau\right|$
Обозначив $z(t) = |y_1(t) - y_2(t)|$, перепишем последнее неравенство следующим образом:

$$0 \leqslant z(t) \leqslant L\left|\int_{t_0}^{t} z(\tau)d\tau\right|,\ t \in [t_1,t_2].$$

Применяя лемму Гронуолла-Беллмана с $c = 0$ и $d = L$, имеем $z(t) = 0,\ t \in [t_1,t_2]$. Следовательно, $y_1(t) = y_2(t),\ t \in [t_1,t_2]$. ■
Теорема (существования) ((локальная)). Пусть функция $f(t,y)$ непрерыв на в Π, удовлетворяет в Π условию Липшица по y и $|f(t,y)| \leq M, (t,y) \in \Pi$. Тогда на отрезке $[t_o - h, t_0 + h]$, где $h = min\{T, \frac{A}{M}\}$, существует функция $y(t)$ такая, что $y(t) \in C^1[t_0-h, t_0+h], |y(t) - y_0| \leq A, t \in [t_0-h, t_0+h]$,
$y'(t) = f(t, y(t)), t \in [t_0-h, t_0+h]\ y(t_0) = y_0$

Следует отметить, что мы можем доказать теорему существования не на всем исходном отрезке $[t0-T, t0+T]$, а на некотором, вообще говоря, меньшем. Поэтому эта теорема часто называется **локальной** *теоремой существования решения задачи Коши.*

[А. М. Денисов, *Обыкновенные дифференциальные уравнения, часть 1*, page 25-30]

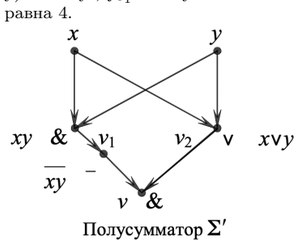
osn 27. Схемы из функциональных элементов и простейшие алгоритмы их синтеза. Оценка сложности схем, получаемых по методу Шеннона.

Орграф – это ориентированный граф.
Вершины орграфа, в которые не входит ни одной дуги, называются **истоками**.
Орграф называется **ациклическим**, если в нем нет ориентированных циклов.
Систему Б $= g_1, g_2, ..., g_m$, где все g_i — функции алгебры логики, будем называть **базисом функциональных элементов**.
Орграф называется **упорядоченным**, если для каждой вершины v_i, в которую входит k_i дуг, задан порядок $e_1, e_2, ..., e_{k_i}$ этих дуг.

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

- каждому истоку приписана некоторая переменная, причем разным истокам приписаны разные переменные (истоки при этом называются входами схемы, а приписанные им переменные — входными переменными);
- каждой вершине, в которую входят k ≥ 1 дуг, приписана функция из базиса Б, зависящая от k переменных(вершина с приписанной функцией при этом называется **функциональным элементом**);
- некоторые вершины выделены как **выходы**.

Сложностью схемы из функциональных элементов называется число функциональных элементов в схеме (число внутренних вершин).
Пример:
Полусумматор Пусть v и v_1 — выходы на рисунке, f_v = x∧y ∧ (x ∨ y)= x + y ; f_{v_1} = x∧y . Сложность (число элементов) полусумматора равна 4.



Полусумматор Σ'

Важные ФАЛ и системы ФАЛ:

- Мультиплексорная ФАЛ μ_n порядка n

$$\mu(\underbrace{x_1,\dots,x_n}_{\text{адресные}}, \underbrace{y_0,\dots,y_{2^n-1}}_{\text{информационные}}) = \bigvee_{\alpha=(\alpha_1,\dots,\alpha_n)} x_1^{\alpha_1}\dots x_n^{\alpha_n} y_{\nu(\alpha)},$$

где $\nu(\alpha)$ – перевод двоичного числа α в десятичное.
Реализуется мультиплексором.

- Универсальная система $\vec{P}_2(n)$ порядка n — содержит все ФАЛ от n переменных. Реализуется универсальным многополюсником.

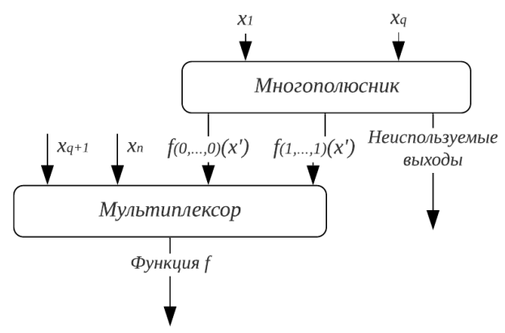
Лемма. Для каждого натурального n существует СФЭ над базисом B $U_n \in U_B^C$(множество всех схем на базисом B), которая реализует систему ФАЛ $\vec{P}_2(n)$ и сложность которой равна $2^{2^n} - n$.
▲ В силу полноты базиса в U_B^C существует система СФЭ Σ от БП $x_1,\dots,x_n$, реализующая систему ФАЛ $\vec{P}_2(n)$. Искомая СФЭ U_n является строго приведённой СФЭ, которая эквивалентна Σ и получается из неё в результате операций присоединения эквивалентных вершин и удаления висячих вершин. Действительно, из построения следует, что число всех вершин СФЭ U_n, включая n её входов, равно 2^{2^n} и поэтому $L(U_n) = 2^{2^n} - n$ (вычитаем n входов, которые автоматически реализуют функции $x_1,\dots,x_n$). ■
Следствие. $L_B^C(\vec{P}_2(n)) \leqslant 2^{2^n} - n$.

Базис $B_0 = \{\&, \vee, \neg\}$

Определения сложности.
Сложность ФАЛ f: $L_B(f) = \min\limits_{\substack{\text{СФЭ } \Sigma \in U_B^C \\ \text{реализующие } f}} L(\Sigma)$
Функция Шеннона: $L_B(n) = \max\limits_{f \in P_2(n)} L_B(f)$

Синтез по совершенной ДНФ
Совершенная ДНФ $f(x_1,\dots,x_n) = \bigvee\limits_{\sigma \in N_f} x_1^{\sigma_1}\&\dots\&x_n^{\sigma_n}$, где N_f — все наборы σ, на которых $f(\sigma) = 1$.
Совершенная ДНФ — формула в B_0, значит существует СФЭ Σ_f над базисом B_0, которая реализует f.
Тогда $L(\Sigma_f) \leqslant \underbrace{2^n}_{1}(\underbrace{n-1}_{2} + \underbrace{n}_{3}) + \underbrace{2^n - 1}_{4}$, где 1 — верхняя оценка $|N_f|$, т.е. количества дизъюнктов в совершенной ДНФ, 2 — количество конъюнкций в каждом дизъюнкте, 3 — верхняя оценка количества отрицаний в дизъюнкте, 4 — оценка количества дизъюнкций между дизъюнктами. СФЭ — частный случай квазидеревьев, которые эквивалентны формулам, поэтому $L^C(n) \leqslant L^\Phi(n)$ Так получаем верхнюю оценку функции Шеннона: $L^C(n) \leqslant L(\Sigma_f) \leqslant n \cdot 2^{n+1}$.

Метод Шеннона.
Выбираем параметр q, $1 \leqslant q \leqslant n$.
Используется **разложение Шеннона**:
$f(\underbrace{x_1,\dots,x_q}_{x'}, \underbrace{x_{q+1},\dots,x_n}_{x''}) =$
$= \bigvee\limits_{\sigma''=(\sigma_{q+1},\dots,\sigma_n)} x_{q+1}^{\sigma_{q+1}}\dots x_n^{\sigma_n} \cdot f_{\sigma''}(x_1,\dots,x_q,\sigma_{q+1},\dots,\sigma_n)$
Для любой ФАЛ $f \in P_2(n)$ строим СФЭ Σ_f как суперпозицию $\Sigma''(\Sigma')$, где Σ'' — мультиплексор, Σ' — универсальный многополюсник.



Сложность многополюсника от q переменных: $L(\Sigma') \leqslant 2^{2^q} - q$ (следует из леммы)
Сложность мультиплексора от $n-q$ переменных: $L(\Sigma'') \leqslant 2^{n-q+2} - 3$
Полагая $q = \lfloor \log_2(n - 2\log_2 n) \rfloor$ получаем в результате преобразований:

$$L(\Sigma_f) \leqslant 2^{2^q} + 4 \cdot 2^{n-q} \leqslant \frac{8 \cdot 2^n}{n - 2\log n} + O(\frac{2^n}{n^2})$$

Таким образом, верна оценка сложности СФЭ: $L^C(n) \lesssim 8 \cdot \frac{2^n}{n}$

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

osn 29. 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**.
$\psi = \int_a^b f(x)dx - \sum_{k=0}^{N} C_k f(x_k)$ — **погрешность квадратурной формулы**.
Введем на $[a,b]$ равномерную сетку $\omega_n = \{x_i = a + ih,\ i \in [0,N],\ N_h = b-a\}$.
$\int_a^b f(x)dx = \sum_{i=0}^{N} \int_{x_{i-1}}^{x_i} f(x)dx$. Строим квадратурные формулы для $\int_{x_{i-1}}^{x_i} f(x)dx$.
Формула прямоугольников.

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

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

$$\int_{x_{i-1}}^{x_i} \left(f(x_{i-\frac{1}{2}}) + (x - x_{i-\frac{1}{2}})f'(x_{i-\frac{1}{2}}) + \frac{\left(x - x_{i-\frac{1}{2}}\right)^2}{2} f''(\xi)\Big|_{\xi \in [x_{i-1};x_i]} - f(x_{i-\frac{1}{2}})\right)dx$$

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

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

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

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 по узлам $x_{i-1},\ x_i$.

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

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

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

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

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

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

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

Формула Симпсона (парабол).
$L_n = \sum_{k=0}^{n} L_{n_k} = \sum_{k=0}^{n} \frac{\omega(x)}{(x - x_k)\omega'(x_k)} f(x_k)$ — интерполяционный полином в Форме Лагранжа, где

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

В формуле Симпсона:

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

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

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

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

$$\int_a^b L_2(x)dx \approx \frac{h}{6}(f_0 + f_N + 2(f_1 + \dots + f_{N-1}) + 4(f_{\frac{1}{2}} + \dots + f_{N-\frac{1}{2}}))$$

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

сюда бы картинки разбиенитя площади под кривой для разных методов

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

osn 31. Численное решение задачи Коши для обыкновенных дифференциальных уравнений. Примеры методов Рунге-Кутта.

Рассматривается задача Коши для системы ОДУ:

$$\begin{cases} \dfrac{du}{dt} = f(t, u(t)), \quad t > 0, \\ u(0) = u_0, \end{cases} \tag{13}$$

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 $f(t,u(t))$ непрерывна в параллелепипеде $R = \{|t| \leqslant a, |u(t) - u(0)| \leqslant b,\ a,b \in \mathbb{R}\}$ и уд-ет в R условию Липшица по второму аргументу, т.е. $|f(t,u) - f(t,v)| \leqslant L|u - v|$, для всех $(t,u),\ (t,v) \in R$ $\implies \exists!$ решение $u(t)$ задачи (13), определенное и непрерывное на некотором отрезке.
В приведенных ниже примерах для простоты изложения предполагается, что система (13) состоит всего из одного уравнения.

Нахождение численного решения

Для нахождения численного решения вводится сетка по времени с постоянным шагом $\tau > 0$, т.е. множество точек $\omega_\tau = \{t_n = n\tau,\ n \in \mathbb{Z}_+\}$, и обозначим $u_n = u(t_n)$, $f_n = f(t_n, u_n)$. Точное решение задачи (13) будем обозначать буквой u, а **приближенное решение** (сеточная ф-ия) — буквой y: $y_n = y_n(t_n)$.
про численное решение

Двухэтапный метод Рунге–Кутта

Общий вид двухэтапного метода Рунге–Кутта для уравнения (13):

$$\begin{cases} \dfrac{y_{n+1} - y_n}{\tau} = \sigma_1 K_1 + \sigma_2 K_2, \quad n \in \mathbb{Z}_+ \\ y_0 = u_0, \\ K_1 = f(t_n, y_n), \quad K_2 = f(t_n + a_2\tau,\ y_n + b_{21}\tau f(t_n, y_n)), \end{cases} \tag{14}$$

где $\sigma_1, \sigma_2, a_2, b_{21} \in \mathbb{R}$ — 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 (14):

$$\frac{y_{n+1} - y_n}{\tau} = \sigma_1 f(t_n, y_n) + \sigma_2 f(t_n + a_2\tau, y_n + b_{21}\tau f(t_n, y_n)).$$

Рассмотрим **погрешность аппроксимации** разностной схемы (14) на решении задачи (13):

$$\psi_n = -\frac{u_{n+1} - u_n}{\tau} + \sigma_1 f(t_n, u_n) + \sigma_2 f\left(t_n + a_2\tau, u_n + b_{21}\tau f(t_n, u_n)\right). \tag{15}$$

Утв.: Погрешность аппроксимации этого метода имеет второй порядок малости по τ: $\psi_n = O(\tau^2)$ при $\sigma_2 = \sigma, \sigma_1 = 1 - \sigma, a_2 = b_{21} = \sigma/2$.
▲
Разложим 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 $f(t_n + a_2\tau, u_n + b_{21}\tau f_n)$ в окрестности точки (t_n, u_n):
$f(t_n + a_2\tau, u_n + b_{21}\tau f(t_n, u_n)) = f(t_n, u_n) + a_2\tau\frac{\partial f_n}{\partial t} + b_{21}\tau f_n\frac{\partial f_n}{\partial u} + O(\tau^2)$
Причем $u'' = \frac{\partial f}{\partial t} + f\frac{\partial f}{\partial u}$. Тогда погрешность аппроксимации принимает вид:
$\psi_n = -u_n' + (\sigma_1 + \sigma_2)f(t_n, u_n) + \tau((a_2\sigma_2 - 0.5)\frac{\partial f_n}{\partial t} + (b_{21}\sigma_2 - 0.5)f_n\frac{\partial f_n}{\partial u} + O(\tau^2)$.
Чтобы получить оценку погрешности со вторым порядком по τ, необходимо избавиться от слагаемых, содержащих τ в первой степени. Для этого положим:
$\sigma_1 + \sigma_2 = 1, \sigma_2 a_2 = \sigma 2 b_{21} = 0.5$
Тогда $\psi_n = O(\tau^2)$.
■
Погрешность решения разностной схемы (14): $z_n = y_n - u_n, n \in \mathbb{Z}$.
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 (13)
Частные случаи общего двухэтапного метода Рунге–Кутта:

1. При $\sigma = 1,\ \ a = a_2 = 0.5,\ \ b = b_{21} = 0.5$ 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:

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

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:

$$K_1 = f(t_n, y_n),$$
$$K_2 = f(t_n + a_2\tau, y_n + b_{21}\tau K_1),$$
$$K_3 = f(t_n + a_3\tau, y_n + b_{31}\tau K_1 + b_{32}\tau K_2),$$
$$\dots$$
$$K_m = f(t_n + a_m\tau, y_n + b_{m1}\tau K_1 + b_{m2}\tau K_2 + \dots + b_{mm-1}\tau K_{m-1}).$$

При этом разностная схема для исходной задачи (13) имеет вид

$$\begin{cases} \dfrac{y_{n+1} - y_n}{\tau} = \sigma_1 K_1 + \sigma_2 K_2 + \dots + \sigma_m K_m \\ y_0 = u_0, n \in \mathbb{Z}_+, \end{cases} \tag{16}$$

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. $m = 3$: $\frac{y_{n+1} - y_n}{\tau} = \frac{1}{6}(K_1 + 4K_2 + K_3)$, 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: $O(\tau^3)$.
2. $m = 4$: $\frac{y_{n+1} - y_n}{\tau} = \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4)$, где
$$K_1 = f(t_n, y_n),$$
$$K_2 = f(t_n + 0.5\tau, y_n + 0.5\tau K_1),$$
$$K_3 = f(t_n + 0.5\tau, y_n + 0.5\tau K_2),$$
$$K_4 = f(t_n + \tau, y_n + \tau K_3).$$
Данная схема имеет четвертый порядок точности по τ: $O(\tau^4)$.

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

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. $a = \sqrt{\dfrac{T_0}{\rho}}$, 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* $\dfrac{\partial^2 u}{\partial t^2} = \dfrac{d^2 u}{dt^2}\ u\ \dfrac{\partial^2 u}{\partial x^2} = \dfrac{d^2 u}{dx^2} : \dfrac{d^2 u}{dt^2} = a^2 \dfrac{d^2 u}{dx^2} \implies d^2 u dx^2 = a^2 dt^2 d^2 u \implies dx^2 = a^2 dt^2$.) Характеристическое уравнение: $dx^2 - a^2 dt^2 = 0$.
$dx - adt = 0,\ dx + adt = 0 \implies x - at = C_1 = const,\ x + at = C_2 = const$
Сделаем замену переменных:

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

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

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

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 $u_{tt} = a^2 u_{xx}$:

$$u_{\xi\xi} \cdot \xi_t^2 + 2u_{\xi\eta} \cdot \xi_t \cdot \eta_t + \underbrace{u_\xi \cdot \xi_{tt}}_{=0} + u_{\eta\eta} \cdot \eta_t^2 + \underbrace{u_\eta \cdot \eta_{tt}}_{=0} =$$

$$= a^2 (u_{\xi\xi} \cdot \xi_x^2 + 2u_{\xi\eta} \cdot \xi_x \cdot \eta_x + \underbrace{u_\xi \cdot \xi_{xx}}_{=0} + u_{\eta\eta} \cdot \eta_x^2 + \underbrace{u_\eta \cdot \eta_{xx}}_{=0})$$

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:

$$u(\xi,\eta) = \int f^*(\eta) d\eta = f_1(\xi) + f_2(\eta) \tag{17}$$

Обратно, каковы бы ни были дважды дифференцируемые функции f_1 и f_2, функция $u(\xi,\eta)$, определяемая формулой (17), представляет собой решение уравнения $u_{\xi\eta} = 0$. Так как всякое решение уравнения $u_{\xi\eta} = 0$ может быть представлено в виде (17) при соответствующем выборе f_1 и f_2, то формула (17) 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 начальным условиям:

$$\begin{cases} u(x,0) = f_1(x) + f_2(x) = \varphi(x) \\ u_t(x,0) = -af_1'(x) + af_2'(x) = \psi(x) \end{cases}$$

Проинтегрируем второе равенство, получим:

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

Сложим и вычтем два полученных равенства:

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

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 математтической физики*, page 50-52]

osn 30. Методы Ньютона и секущих для решения нелинейных уравнений.

👁 ф-ию $f(x),\ x \in \mathbb{R}$, и ур-ие $f(x) = 0$.
❑ $x^* \in \mathbb{R}$ — корень уравнения, и определена его окрестность радиуса a, не содержащая других корней уравнения: $U_a(x^*) = \{x : |x - x^*| < a\}$, причем заданная функция $f(x)$ определена на этой окрестности. Считаем, что начальное приближение $x^0 \in U_a(x^*)$ задано. Тогда для нахождения численного решения уравнения в рассматриваемой окрестности необходимо построить последовательность $\{x^n\}$, сходящуюся к корню x^* уравнения: $\lim\limits_{n\to\infty} f(x^n) = f(x^*) = 0$.
Численное решение нелинейных уравнений можно разбить на 2 этапа:

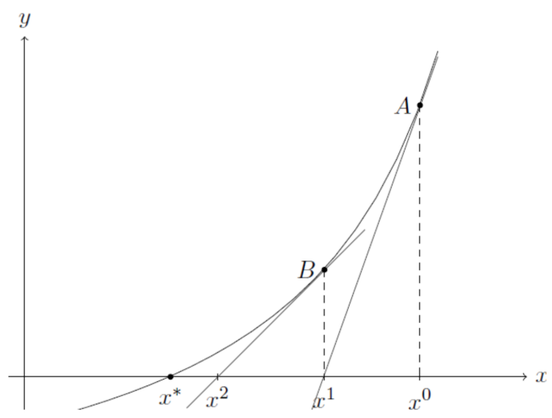
1. Локализация корня, т.е. определение окрестности $U_a(x^*)$.
2. Задание итерационного процесса — построение последовательности $\{x^n\}$, сходящейся к корню уравнения.

Метод Ньютона

❑ в $U_a(x^*)$ существует и не обращается в ноль непрерывная первая производная функции $f(x)$: $f'(x) \neq 0, \quad x \in U_a(x^*)$.
Разложим $f(x^*)$ по формуле Тейлора в малой окрестности точки $x \in U_a(x^*)$: $f(x^*) = f(x) + (x^* - x)f'(x) + \ldots$, и отбросим в этом разложении величины, имеющие второй и выше порядок малости по $(x^* - x)$.
Заменив x^* на x^{n+1} и x на x^n, получим ур-ие $f(x^n) + (x^{n+1} - x^n)f'(x^n) = 0, \quad n \in \mathbb{Z}_+$.
Учитывая, что $f'(x^n) \neq 0$, имеем:

$$x^{n+1} = x^n - \frac{f(x^n)}{f'(x^n)}, \quad n \in \mathbb{Z}_+. \tag{18}$$

Итерационный процесс поиска корня уравнения $f(x) = 0$, задаваемый формулой (18), наз-ся **итерационным методом Ньютона**.



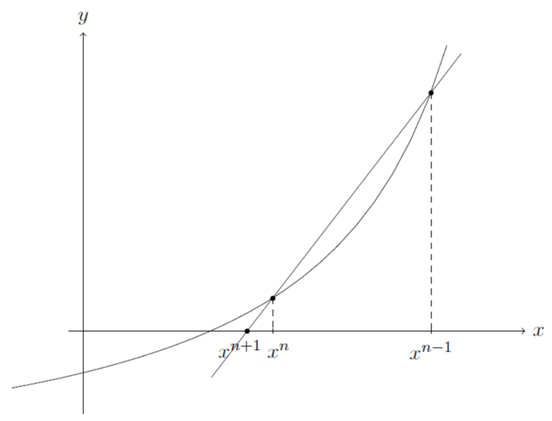
👁 т. $A(x^0, f(x^0))$. Определим первую итерацию x^1 как абсциссу точки пересечения с осью Ox касательной к $f(x)$, проведенной через т. A. Аналогично получаем значение x^2. Продолжая, на n-ом шаге получим значение x^n, приближающее корень x^* уравнения $f(x) = 0$ с заданной точностью.
Зам. При решении задач на практике часто рассматривается модифицированный метод Ньютона, задаваемый формулой $x^{n+1} = x^n - \dfrac{f(x^n)}{f'(x^0)}, \quad n \in \mathbb{Z}_+$ (чтобы считать пр-ую только один раз).

Метод секущих

В методе Ньютона (18) заменим $f'(x)$ на его дискретный аналог $\dfrac{f(x^n) - f(x^{n-1})}{x^n - x^{n-1}}$, получаем итерационный метод:

$$x^{n+1} = x^n - \frac{(x^n - x^{n-1}) f(x^n)}{f(x^n) - f(x^{n-1})} \tag{19}$$

Итерационный процесс 19 задает двухшаговый метод решения уравнений, называемый **методом секущих**.



Сходимость метода Ньютона

Если рассмотреть итерационный метод Ньютона как метод простой итерации ($x^{n+1} = S(x^n), n \in \mathbb{Z}_+$) с функцией $S(x) = x - \dfrac{f(x)}{f'(x)}$.
$|S'(x)| < 1$ при $x \in U_a(x^*)$, то он сходится. Предполагая, что функция $f(x)$ дифференцируема достаточное число раз, продифференцируем функцию $S(x)$:

$$S'(x) = 1 - \frac{(f'(x))^2 - f(x)f''(x)}{(f'(x))^2} = \frac{f(x)f''(x)}{(f'(x))^2}.$$

Теорема: ❑ $\exists$ такая константа $M > 0$, для которой выполнена оценка $\dfrac{1}{2}|S''(x)| \leqslant M, \quad x \in U_a(x^*)$. Тогда если начальное приближение x^0 выбрать в соответствии с условием $|x^0 - x^*| < \dfrac{1}{M}$, то итерационный метод Ньютона сходится, и имеет место оценка: $|x^n - x^*| \leqslant \dfrac{1}{M}\left(M|x^0 - x^*|\right)^{2^n}$.
▲ Погрешность приближенного решения: $z^n = x^n - x^*$.
👁 выражение для z^{n+1}: $z^{n+1} = x^{n+1} - x^* = S(z^n + x^*) - S(x^*)$.
Разложим $S(z^n + x^*)$ по формуле Тейлора и учитывая $S'(x^*) = 0$:

$$z^{n+1} = S(x^*) + S'(x^*)z^n + \frac{1}{2}S''(\tilde{x}^n)(z^n)^2 - S(x^*) = \frac{1}{2}S''(\tilde{x}^n)(z^n)^2,$$

$$\tilde{x}^n = x^n + \theta z^n,\ \theta \in \mathbb{R},\ |\theta| < 1.$$

❑ функция $f(x)$ трижды непрерывно дифференцируема в окрестности $U_a(x^*)$. Тогда $S''(x) = \left(\dfrac{f(x)f''(x)}{(f'(x))^2}\right)'$.
❑ $\exists$ постоянная $M > 0$ такая, что для любого $x \in U_a(x^*)$ выполняется неравенство $M \geqslant \dfrac{1}{2}|S''(x)|$.
Из этого неравенства и уравнения z^{n+1} следует оценка $|z^{n+1}| \leqslant M|(z^n)^2|$.
Домножим это неравенство на M и обозначим $v^n = M|z^n|$. Тогда получим, что $v^{n+1} \leqslant (v^n)^2$.
Отсюда следует, что $v^n \leqslant (v^0)^{2^n}$, значит, $M|z^n| \leqslant \left(M|z^0|\right)^{2^n}$, $|z^n| \leqslant \dfrac{1}{M}\left(M|z^0|\right)^{2^n}$.
Введем обозначение $q = M|z_0|$. Если $0 < q < 1$, то последовательность $\{z^n\}_{n=0}^{\infty}$ стремится к нулю: $z^n \underset{n\to\infty}{\longrightarrow} 0$, и итерационный метод Ньютона сходится. Условие на q $(0 < q < 1)$ будет выполнено, если $0 < |z^0| < \dfrac{1}{M}$, то есть $|x^0 - x^*| < \dfrac{1}{M}$. ■

[Ионкин, *Лекции по курсу Численные методы*, page 99-107]

osn 28. Вероятностное пространство. Случайные величины. Закон больших чисел в форме Чебышева.

Вероятностное пространство

Пространство элементарных исходов Ω – любое непустое множество, содержащее все возможные результаты случайного эксперимента. Элементы $\omega \in \Omega$ – **элементарные исходы** – исходы случайного эксперимента, из которых в эксперименте происходит ровно один.
Множество $\mathcal{F}$, элементами которого являются подмножества множества Ω (не обязательно все) называется **σ-алгеброй** (сигма-алгеброй), если выполнены следующие условия:
1. $\Omega \in \mathcal{F}$;
2. если $A \in \mathcal{F}$, то $\neg A \in \mathcal{F}$;
3. $A_1, A_2, \ldots \in \mathcal{F} \implies \bigcup\limits_{i=1}^{\infty} A_i \in \mathcal{F}$ (объединение по *счетному* числу подмножеств)

Минимальная σ-алгебра, содержащая множество всех интервалов на вещественной прямой, называется **борелевской σ-алгеброй** в $\mathbb{R}$ и обозначается $\mathcal{B}(\mathbb{R})$.
Борелевское множество — элемент борелевской σ-алгебры.
Вероятность P — действительная функция случайного события: $\mathcal{F} \to \mathbb{R}$, удовлетворяющая следующим условиям:
1. $P(\Omega) = 1$;
2. $\forall A \in \mathcal{F} \implies P(A) \geqslant 0$;
3. $\forall A_1, A_2, \ldots, A_n, \cdots \in \mathcal{F} : \ A_i \cap A_j = \varnothing (i \neq j) \implies P(\bigcup\limits_{i=1}^{\infty} A_i) = \sum\limits_{i=1}^{\infty} P(A_i)$ (счётная аддитивность).

Вероятностное пространство – тройка $(\Omega, \mathcal{F}, P)$, где Ω – множество элементарных исходов, $\mathcal{F}$ – σ-алгебра над Ω, вероятность P определена на $\mathcal{F}$.
Пример: Пусть $\Omega = (\omega_1, \ldots, \omega_s)$, $\mathcal{F}$ — всевозможные подмножества множества Ω.
$P(\omega_1) = \cdots = P(\omega_s) = \dfrac{1}{s} \implies P(A) = \dfrac{|A|}{|\Omega|}$ — классическое определение вероятности.

Случайные величины

Пара $(X, \mathcal{U})$, где X – произвольное множество, а $\mathcal{U}$ – σ-алгебра над ним – **измеримое пространство**.
Например, $(\Omega, \mathcal{F})$ и $(R, \mathcal{B})$ – измеримые пространства.
Элементы σ-алгебры $\mathcal{U}$ называются **измеримыми множествами**.
Пусть даны измеримые пространства $(\Omega, \mathcal{F})$ и $(R, \mathcal{B})$. Функция $\xi : \Omega \to R$ называется **случайной величиной**, если прообраз любого борелевского множества $B \in \mathcal{B}$ является событием.
Распределением случайной величины ξ называется функция $P_\xi : \ \mathcal{B} \to \mathbb{R}$, определенная $\forall B \in \mathcal{B}$ по правилу $P_\xi(B) = P(\xi \in B)$.
Функцией распределения случайной величины ξ называется отображение $F_\xi : \ \mathbb{R} \to \mathbb{R}$, определённое по правилу: $F_\xi(x) = P(\xi < x)$ или $F_\xi(x) = P_\xi((-\infty, x))$.
Случайная величина ξ называется **дискретной случайной величиной**, если существует не более чем счетное множество B такое, что $P_\xi(B) = 1$. Распределение, соответствующее дискретной случайной величине, также называется дискретным.
Дискретная функция распределения имеет вид:

$$F_\xi(x) = P(\xi < x) = \sum_{x_i < x} p_i = \sum_{x_i < x} P(\xi = x_i).$$

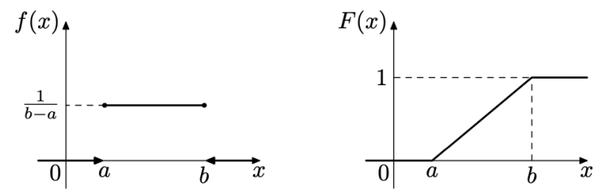
Распределение ξ называется **абсолютно непрерывным**, если существует $f(x) \geqslant 0$ такая, что для любого борелевского множества B справедливо

$$P_\xi(B) = \int_B f(x)\lambda(dx),$$

где $f(x)$ — *плотность распределения*, λ — мера Лебега. *Абсолютно непрерывная функция распределения* имеет вид:

$$F_\xi(x) = P(\xi < x) = \int_{-\infty}^{x} f(t)dt.$$

Пример: равномерное распределение $f(x) = \begin{cases} \dfrac{1}{b-a}, & x \in [a,b] \\ 0, & x \notin [a,b] \end{cases}$



Математическим ожиданием случайной величины ξ называется число $E\xi = \int\limits_\Omega \xi(\omega)P(d\omega)$ или $\int\limits_{-\infty}^{+\infty} x dF_\xi(x)$. Если интеграл расходится, то говорят, что математического ожидания не существует.
В случае дискретной сл. вел.: $E\xi = \sum\limits_i x_i p_i$.

В случае абсолютно непрерывной сл. вел.: $E\xi = \int x f(x) dx$

Дисперсией случайной величины ξ называется число $D\xi = E(\xi - E\xi)^2$, $\sigma = \sqrt{D\xi}$ – **среднеквадратическое отклонение**.
Величина $\operatorname{cov}(\xi,\eta) \equiv E(\xi\eta) - E\xi\, E\eta$ называется **ковариацией** случайных величин ξ и η. Если ξ и η независимы, то $\operatorname{cov}(\xi,\eta) = 0$. Обратное, вообще говоря, неверно.

Закон больших чисел в форме Чебышева

Неравенство Чебышёва Если $\exists D\xi$, то $\forall \varepsilon > 0$

$$P(|\xi - E\xi| \geqslant \varepsilon) \leqslant \frac{D\xi}{\varepsilon^2}.$$

▲ Для $\varepsilon > 0$ неравенство $|\xi - E\xi| \geqslant \varepsilon \iff (\xi - E\xi)^2 \geqslant \varepsilon^2$, поэтому $P(|\xi - E\xi| \geqslant \varepsilon) = P\left((\xi - E\xi)^2 \geqslant \varepsilon^2\right) \leqslant \dfrac{E(\xi - E\xi)^2}{\varepsilon^2} = \dfrac{D\xi}{\varepsilon^2}$. ■
Следствие: Если $\varepsilon = 3\sigma$, где σ — стандартное отклонение, то получим $P(|\xi - E\xi| > 3\sigma) \leqslant \dfrac{D\xi}{9\sigma^2} = \dfrac{D\xi}{9\,D\xi} = \dfrac{1}{9} \iff P(|\xi - E\xi| \leqslant 3\sigma) \geqslant 1 - \dfrac{1}{9} = \dfrac{8}{9}$.
Это соотношение называется **правилом трёх сигм**.
Последовательность случайных величин $\{\xi_n\}_{n=1}^{+\infty}$ **сходится по вероятности** к случайной величине ξ ($\xi_n \xrightarrow{P} \xi$), если

$$\forall \varepsilon > 0 \quad P\left(\left\{\omega\colon |\xi_n(\omega) - \xi(\omega)| > \varepsilon\right\}\right) \xrightarrow[n\to+\infty]{} 0.$$

Закон больших чисел в форме Чебышёва: $\forall$ последовательности $\xi_1, \xi_2, \ldots$ попарно независимых и одинаково распределённых случайных величин для которых $E\xi_i^2 < \infty \implies$

$$\frac{\xi_1 + \ldots + \xi_n}{n} \xrightarrow[n\to+\infty]{P} E\xi_1.$$

▲ $S_n = \xi_1 + \ldots + \xi_n$, Из линейности матожидания получим

$$E\left(\frac{S_n}{n}\right) = \frac{E\xi_1 + \ldots + E\xi_n}{n} = \frac{nE\xi_1}{n} = E\xi_1.$$

Пусть $\varepsilon > 0$. Воспользуемся неравенством Чебышёва:

$$P\left(\left|\frac{S_n}{n} - E\left(\frac{S_n}{n}\right)\right| \geqslant \varepsilon\right) \leqslant \frac{D\left(\frac{S_n}{n}\right)}{\varepsilon^2} = \frac{DS_n}{n^2\varepsilon^2} = \frac{D\xi_1 + \ldots + D\xi_n}{n^2\varepsilon^2} =$$

$$= \frac{nD\xi_1}{n^2\varepsilon^2} = \frac{D\xi_1}{n\varepsilon^2} \xrightarrow[n\to+\infty]{} 0,$$

т.к. $D\xi_1 < \infty$. Дисперсия суммы превратилась в сумму дисперсий в силу попарной независимости слагаемых, из-за которой все ковариации $\operatorname{cov}(\xi_i, \xi_j)$ по свойству ковариации обратились в нуль при $i \neq j$. ■
Центральная предельная теорема: Пусть $\xi_1, \xi_2, \ldots$ — последовательность независимых одинаково распределенных невырожденных случайных величин с $E\xi_1^2 < \infty$ и $S_n = \xi_1 + \ldots + \xi_n$. Тогда

$$P\left(\frac{S_n - ES_n}{\sqrt{DS_n}} \leqslant x\right) \xrightarrow[n\to+\infty]{} \Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{u^2}{2}} du \quad \forall x \in R$$

▲ Пусть $E\xi_1 = m$, $D\xi_1 = \sigma^2$. Введём $X = \xi_1 - m$ и $D\phi_X(t) = Ee^{itX}$ и $D\phi_n(t) = Ee^{it\frac{S_n - ES_n}{\sqrt{DS_n}}} = \left[D\phi_X\left(\dfrac{t}{\sigma\sqrt{n}}\right)\right]^n$
В силу разложения характеристической функции (при $\exists$ соответствующих моментов) $D\phi_X(t) = 1 + itEX + \ldots + \dfrac{(it)^n}{n!}EX^n + R_n(t)$
Т.к. $EX = E[\xi_1 - m] = 0$, при $n = 2$: $D\phi(t) = 1 - \dfrac{\sigma^2 t^2}{2} + \overline{o}(t^2), \quad t \to 0$.
$\forall t \in R$ при $n \to +\infty \implies D\phi_n(t) = \left[1 - \dfrac{\sigma^2 t^2}{2\sigma^2 n} + \overline{o}\left(\dfrac{1}{n}\right)\right]^n \to e^{-\frac{t^2}{2}}$
Функция $e^{-\frac{t^2}{2}}$ – характеристическая функциия $N_{0,1}$. В силу теоремы о непрерывном соответствии между функциями распределения и характеристическими функциями Ц.П. теорема доказана. ■

[И. С. Рожков, *Почти наверное достаточное учебное пособие. Теория вероятностей и математическая статистика. Статбук*, page 1-53]

osn 26. Функции алгебры логики. Реализация их формулами. Совершенная дизъюнктивная нормальная форма.

Пусть $E_2 = 0{,}1$ - основное множество. Тогда $E_2^n = \{(a_1, a_2, \ldots, a_n)\} \mid \forall i, a_i \in E_2\}$. Тогда всюду определенной булевой функцией назовем отображение $f_n(x_1, x_2, \ldots, x_n)$: $E_2^n \to E_2$. Такую функцию можно задать таблично, а можно как суперпозицию других, более простых функций. Например, для n=1:

x	0	1	x	$\overline{x}$
0	0	1	0	1
1	0	1	1	0

При этом функция 0 называется константным нулем, функция 1 - константной единицей, функция x - тождественной, а функция $\overline{x}$ - отрицанием x.
Для n = 2:

x	y	f_1	f_2	f_3	f_4	f_5	f_6	f_7
0	0	0	0	0	1	1	1	1
0	1	1	0	1	1	0	1	0
1	0	1	0	1	0	0	1	0
1	1	1	1	0	1	1	0	0

При этом: f_1 - дизъюнкция $f_1 = x \vee y$
f_2 - конъюнкция $f_2 = x \wedge y$
f_3 - сложение по модулю два $f_3 = x \oplus y$
f_4 - импликация $f_4 = x \to y$
f_5 - эквивалентность $f_5 = x \equiv y$
f_6 - штрих Шеффера $f_6 = x \mid y$
f_7 - стрелка Пирса $f_7 = x \downarrow y$

Лемма (о числе слов): В алфавите $A = \{a_1, a_2, \ldots, a_r\}$ из r букв можно построить ровно r^m различных слов длины m.
▲ Проведем индукцию по m. Для $m = 1$ утверждение очевидно. Пусть утверждение леммы верно для $m - 1$, то есть существует ровно $r^{(m-1)}$ различных слов длины $m - 1$. Для каждого такого слова длины $m - 1$ существует ровно r возможностей добавить одну букву в конец. Так как всего слов длины $m - 1$ — $r^{(m-1)}$, то различных слов длины m получится $r^* r^{(m-1)} = r^m$. ■
👁 таблицу некоторой функции алгебры логики от n переменных:

x_1	x_2	...	x_n	f
0	0	...	0	a_1
0	0	...	1	a_2
...	...	...	...	...
1	1	...	1	$a_{2_n - 1}$

Для ее задания необходимо и достаточно определить ее значения на 2^n наборах. Так, получим, что всего различных функций от n переменных столько, сколько существует различных наборов из нулей и единиц длины 2^n, то есть 2^{2^n}.

Переменная x_i называется **существенной** переменной функции алгебры логики $f_n(x_1, x_2, \ldots, x_n)$, если существуют такие $(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n)$, что
$f_n(a_1, \ldots, a_{i-1}, 0, a_{i+1}, \ldots, a_n) \neq f_n(a_1, \ldots, a_{i-1}, 1, a_{i+1}, \ldots, a_n)$
Такие наборы, отличающиеся лишь одной переменной x_i, называются **соседними** по x_i. В противном случае переменная x_i называется **фиктивной**.
Если x_i - фиктивная переменная функции f, то функция f однозначно определяется некоторой функцией $g_n(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)$. Таблицу любой функции можно расширить путем введения новых фиктивных переменных. Две функции алгебры логики называются равными, если одну из них можно получить путем добавления и изъятия любого числа фиктивных переменных. Пусть имеется некоторое множество функций:
$A = \{f_1(\ldots), f_2(\ldots), \ldots, f_n(\ldots)\}$
Введем понятие **формулы** над A:

1. Любая функция из A называется формулой над A.
2. Если $f(x_1, x_2, ..., x_n) \in A$ и для любого iH_i - либо переменная, либо формула над A, то выражение вида f(H_1, H_2, ..., H_n) - формула над A.
3. Только те объекты называются формулами над A, которые можно построить с помощью пунктов 1 и 2.

Основные эквивалентности:

• Коммутативность:
$x \vee y = y \vee x$
$yx = xy$
$x + y = y + x$
$x \equiv y = y \equiv x$

• Ассоциативность
$(x \vee y) \vee z = x \vee (y \vee z) = x \vee y \vee z$
$xyz = (xy)z = x(yz)$
$x + y + z = (x + y) + z = x + (y + z)$

• Дистрибутивность
$(x + y) \wedge z = (x \wedge z) + (y \wedge z)$
$(x \vee y) \wedge z = (x \wedge z) \vee (y \wedge z)$
$(x \wedge y) \vee z = (x \vee z) \wedge (y \vee z)$

• Закон снятия двойного отрицания
$\overline{\overline{x}} = x$

• Правила де Моргана:
$\overline{x \vee y} = \overline{x} \wedge \overline{y}$
$\overline{x \wedge y} = \overline{x} \vee \overline{y}$

• Законы поглощения:
$x \vee x = x$
$x \wedge x = x$
$x \vee \overline{x} = 1$
$x \wedge \overline{x} = 0$
$x \vee 1 = 1$
$x \wedge 1 = x$
$x \vee 0 = x$
$x \wedge 0 = 0$

• остальные формулы:
$x \mid y = \overline{xy}$
$x \downarrow y = \overline{x \vee y}$
$x \to y = \overline{x} \vee y$
$x + y = (x \wedge \overline{y}) \vee (y \wedge \overline{x})$
$x \sim y = \overline{x + y} = (xy) \vee (\overline{xy})$

Приоритет конъюнкции выше, чем приоритеты дизъюнкции и суммы по модулю два.
x в степени σ называется функция $x^\sigma = x$, если $\sigma = 1$; $x^\sigma = \overline{x}, \sigma = 0$.

Теорема о разложении функции алгебры логики по переменным: $\forall$ функции алгебры логики $f(x_1, x_2, \ldots, x_n)$ и $\forall k \in [1, n]$ справедливо следующее равенство:
$f(x_1, x_2, \ldots, x_n) =$
$= \bigvee\limits_{(\sigma_1, \ldots, \sigma_k) \in E^k} x_1^{\sigma_1} x_2^{\sigma_2} \ldots x_k^{\sigma_k} f(\sigma_1, \sigma_2, \ldots, \sigma_k, x_{k+1}, x_{k+2}, \ldots, x_n)$
▲ $\forall$ набора $a = (a_1, \ldots, a_n)$ вычислим значение правой части на этом наборе. Как только один из сомножителей будет равен 0, вся конъюнкция обратится в 0. Таким образом, из ненулевых конъюнкций останется лишь одна — та, в которой $a_i = \sigma_i$ и
$\bigvee a_1^{\sigma_1} a_2^{\sigma_2} \ldots a_n^{\sigma_n} f(\sigma_1, \sigma_2, \ldots, \sigma_k, a_{k+1}, a_{k+2}, \ldots, a_n) = 0 \vee 0 \vee \ldots \vee 0 \vee a_1^{a_1} a_2^{a_2} \ldots a_n^{a_n} f(a_1, a_2, \ldots, a_n)$
Так как $x^x = 1$, указанное выражение равно $f(a_1, \ldots, a_n)$. ■
Литерал – это переменная или отрицание переменной.
Конъюнкция – любое произведение нескольких литералов, не содержащее одинаковых и противоположных литералов.
Дизъюнктивная нормальная форма (ДНФ) – дизъюнкция нескольких различных конъюнкций (одна отдельная конъюнкция – тоже ДНФ).
Совершенная дизъюнктивная нормальная форма (СДНФ) – ДНФ, в каждой конъюнкции которой есть литерал каждой переменной.

Теорема о совершенной дизъюнктивной нормальной форме: $\forall$ функции алгебры логики $f(x_1, x_2, \ldots, x_n)$, отличной от тождественного нуля, справедливо следующее представление:
$f(x_1, x_2, \ldots, x_n) = \bigvee x_1^{\sigma_1} x_2^{\sigma_2} \ldots x_n^{\sigma_n}$
▲ Пусть $f(x_1, x_2, \ldots, x_n)$ отлична от тождественного нуля. Разложим данную функция по $k = n$ переменным:
$f(x_1, x_2, \ldots, x_n) = \bigvee x_1^{\sigma_1} x_2^{\sigma_2} \ldots x_n^{\sigma_n} f(\sigma_1, \sigma_2, \ldots, \sigma_n)$
при этом $(\sigma_1, \sigma_2, \ldots, \sigma_n) \in E_2^n$.

Это может быть переписано в эквивалентном виде:
$f(x_1, x_2, \ldots, x_n) = \bigvee\limits_{(\sigma_1, \sigma_2, \ldots, \sigma_n) : f(\sigma_1, \sigma_2, \ldots, \sigma_n) = 1} x_1^{\sigma_1} \ldots x_n^{\sigma_n}$
Это представление называется **СДНФ** ■

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

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

Полная система (в P_2) — множество $\mathcal{A}$ ФАЛ такое, что любую ФАЛ можно выразить формулой над $\mathcal{A}$.
$\beth$ $\mathcal{A} \subseteq P_2$. Тогда **замыкание** $\mathcal{A}$ (обозн. $[\mathcal{A}]$) — множество всех ФАЛ, которые можно выразить формулами над $\mathcal{A}$.
Класс $\mathcal{A}$ называется **замкнутым**, если $\mathcal{A} = [\mathcal{A}]$.
Говорят, что набор $\tilde{\alpha} = (\alpha_1, ...\alpha_n)$ **предшествует** набору $\tilde{\beta} = (\beta_1, ...\beta_n)$, $\tilde{\alpha} \preccurlyeq \tilde{\beta}$ ($\tilde{\alpha}$ не больше $\tilde{\beta}$), если $\alpha_i \leqslant \beta_i$ для всех $i = 1, ..., n$.
Если $\tilde{\alpha} \preccurlyeq \tilde{\beta}$ и $\tilde{\alpha} \neq \tilde{\beta}$, то говорят, что набор $\tilde{\alpha}$ **строго предшествует** набору $\tilde{\beta}$, $\tilde{\alpha} \prec \tilde{\beta}$.
Наборы $\tilde{\alpha}$ и $\tilde{\beta}$ называются сравнимыми, если $\tilde{\alpha} \preccurlyeq \tilde{\beta}$ либо $\tilde{\beta} \preccurlyeq \tilde{\alpha}$.
Например, вектора $[0, 1]$ 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

– $L = \{f(x_1, \dots, x_n) = a_0 \oplus a_1 x_1 \oplus \dots \oplus a_n x_n\}$ — линейные функции;

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

– $M = \{\alpha \leqslant \beta \to f(\alpha) \leqslant f(\beta)\}$ — монотонные функции;

Вспомогательные леммы:

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

   ▲ Немонотонна $\implies$ $\exists$ два набора $\alpha < \beta$, а $f(\alpha) = 1 > f(\beta) = 0$. Будем заменять в α 0 на 1 по одному, чтобы получился β, промежуточные назовем α_k, $\alpha_0 = \alpha, \alpha_r = \beta$. В какой-то момент получим $f(\alpha_k) = 1, f(\alpha_{k+1} = 0)$, получили 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 $\tilde{\alpha}_k$. $f(\tilde{\alpha}_k) = \bar{x}$ ■

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

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

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

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

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

   $\phi(x, y) = f(x \oplus b, y \oplus a, \alpha_3, \dots, \alpha_n) = (x \oplus b)(y \oplus a) \oplus a(x \oplus b) \oplus b(y \oplus a) \oplus c =$ {раскройте сами} $xy \oplus d$, d-константа. ■

Теорма Поста. Система ФАЛ $\mathcal{A} = \{f_1, f_2, \dots\}$ является полной в P_2 $\iff$ она не содержится целиком ни в одном из следующих классов: T_0, T_1, S, L, M.
▲ *Необходимость.* Пусть $\mathcal{A}$ — полная система, N — любой из классов T_0, T_1, S, L, M , и (от противного) пусть $\mathcal{A} \subseteq N$. Тогда $[\mathcal{A}] \subseteq [N] = N \neq P_2$, то есть $[\mathcal{A}] \neq P_2$. Полученное противоречие завершает обоснование необходимости.
Достаточность. Пусть $\mathcal{A}$ не является подмножеством ни одного из этих классов. Тогда в $\mathcal{A}$ 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 $\mathcal{B} = \{f_0, f_1, f_M, f_L, f_S\}$ Достаточно показать, что $[\mathcal{A}] \supseteq [\mathcal{B}] = P_2$.
Выразим формулами над $\mathcal{B}$ все функции из полной системы $\{0, 1, \overline{x}, x \cdot y\}$.

• *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 сохраняет нуль, $\varphi_0(0) = f_0(0, 0, \dots, 0) = 1$. 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 $\varphi_1 = \bar{x}$, либо $\varphi_1(x) \equiv 0$.

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 функции 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 нелинейной 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: $[f_L, 0, 1, \bar{x}] \supseteq \{x \cdot y, \overline{x \cdot y}\}$.

Конъюнкция получена.

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

[*Презентации к госам от маткиба*]

dop 3. Логика 1-го порядка. Выполнимость и общезначимость. Общая схема метода резолюций.

Базовые символы:

- Предметные переменные $Var = \{x_1, x_2, \dots, x_k, \dots\}$;
- Предметные константы $Const = \{c_1, c_2, \dots, c_l, \dots\}$;
- 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 $\langle Const, Pred, Func \rangle$ называется **сигнатурой алфавита**.
Логические связки и кванторы:
Конъюнкция — $\vee$
Дизъюнкция — $\wedge$
Отрицание — $\neg$
Импликация — $\longrightarrow$
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.
$Term$ — множество термов заданного алфавита. Var_t — множество переменных, входящих в состав терма t.
$t(x_1, x_2, \dots, x_n)$ — 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 $\bar{\phi}$, $\psi \vee \phi$ 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 являющееся связанным, называется **свободным**. Переменная называется **свободной**, если она имеет свободное вхождение в формулу. $\psi(x_1, x_2, \dots, x_n)$ — запись, обозначающая формулу ψ, у которой $Var_\psi \supset \{x_1, x_2, ..., x_n\}$. Если $Var_\psi = 0$, то формула ψ называется замкнутой формулой , или предложением . $CForm$ — множество всех замкнутых формул.
Интерпретация сигнатуры — это $\langle D_I, \overline{Const}, \overline{Func}, \overline{Pred} \rangle$, где

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

• $\overline{Const}$ — оценка констант — сопоставляет каждой константе предмет из области интерпретации.

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

• $\overline{Pred}$ — оценка предикатных символов — сопоставляет каждому предикатному символу отношение на области интерпретации.

Отношение 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 $\overline{P}\,(t_1[d_1, \dots, d_n], \dots, t_n[d_1, \dots, d_n]) = True$,
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 $\psi_1 \to \psi_2$ 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 $\Gamma \models \psi$ обозначает, что ψ — логическое следствие Γ.
Теорема о логическом следствии. Пусть $\Gamma = \{\psi_1, \dots, \psi_n\} \subseteq CForm$, $\phi \in CForm$. Тогда $\Gamma \models \phi \iff \models \psi_1 \wedge \dots \wedge \psi_n \longrightarrow \phi$.
▲ ($\Rightarrow$) Пусть I — произвольная интерпретация. Если $I \not\models \psi_1 \wedge \dots \wedge \psi_n$, то $I \dots \psi_1 \wedge \dots \wedge \psi_n \longrightarrow \phi$.
Если $I \models \psi_1 \wedge \dots \wedge \psi_n$, то $I \models \psi_i$, $i \in [1, n]$, т. е. I — модель для Γ. Поскольку $\Gamma \models \phi$, получаем $I \models \phi$.
Значит, $I \models \psi_1 \wedge \dots \wedge \psi_n \longrightarrow \phi$. Таким образом, для любой интерпретации I имеет место $I \models \psi_1 \wedge \dots \wedge \psi_n \longrightarrow \phi$. Значит, $\psi_1 \wedge \dots \wedge \psi_n \longrightarrow \phi$ — общезначимая формула.
($\Leftarrow$) Пусть I — модель для множества предложений Γ, т. е. $I \models \psi_i$, $i \in [1, n]$. Тогда $I \models \psi_1 \wedge \dots \wedge \psi_n$. Поскольку $\psi_1 \wedge \dots \wedge \psi_n \longrightarrow \phi$ - общезначимая формула, имеет место $I \models \psi_1 \wedge \dots \wedge \psi_n \longrightarrow \phi$. Значит $I \models \phi$. Так как I — произвольная 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, преобразуясь при этом из одной формы в другую. Практически важно уметь определять эти каналы и настраивать их на извлечение нужных знаний.

• База знаний — множество предложений Γ;

• Запрос к базе знаний — предложение ϕ;

• Получение ответа на запрос — проверка логического следствия $\Gamma \models \phi$.

Метод резолюции
Подстановка θ – **унификатор** выражений E_1, E_2, если $E_1\theta = E_2\theta$.
Подстановка θ – **наиболее общий унификатор (НОУ)** выражений E_1, E_2, если θ – унификатор выражений E_1, E_2 и для любого унификатора η существует такая подстановка ρ, для которой верно $\eta = \theta\rho$.
Метод резолюций — метод проверки формулы φ на общезначимость.
Общая схема метода:

1. Свести проблему общезначимости к проблеме противоречивости: если формула общезначима, то ее отрицание невыполнимо: $\psi = \neg\varphi$
2. Построить предварённую нормальную форму (ПНФ): $\psi = Q_1 x_1 \dots Q_n x_n (D_1 \wedge \dots \wedge D_N)$, где Q_i либо квантор существования, либо квантор всеобщности, а D_i — дизъюнкт из КНФ.
3. Построить сколемовскую стандартную форму (ССФ): $\forall x_{i_1} \dots \forall x_{i_k} (D_1 \wedge \dots \wedge D_N)$. При этом формулы будут неэквивалентны, однако свойство противоречивости сохранится.
4. Построить систему дизъюнктов: $S_\varphi\{D_1, \dots, D_N\}$
5. Резолютивно (с использованием правил резолюции и склейки) вывести из системы S_φ пустой дизъюнкт, тем самым доказав противоречивость.

   • Правило резолюции: $\dfrac{D_1' \vee L_1,\ D_2' \vee \neg L_2}{(D_1' \vee D_2')\theta},\ \theta \in \text{НОУ}(L_1, L_2)$

   • Правило склейки: $\dfrac{D_1' \vee L_1 \vee L_2}{(D_1' \vee L_1)\eta},\ \eta \in \text{НОУ}(L_1, L_2)$

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

dop 5. Сортировка. Простейшие алгоритмы — сортировка выбором, вставками, обменом. Оценка сложности алгоритмов сортировки. Быстрая сортировка и ее сложность в среднем и в наихудшем случаях.

Задача
👁 массив $x_1, \dots, x_n$. Задача: переставить элементы массива так, чтобы элементы оказались упорядоченными: $x_1 \leqslant x_2 \leqslant \dots \leqslant x_n$.
Сортировка выбором
Проходим по массиву в поисках наименьшего элемента. Найденный минимум меняем местами с последним элементом. Неотсортированная часть массива уменьшилась на один элемент (не включает первый элемент, куда мы переставили найденный минимум). К этой неотсортированной части применяем те же действия — находим минимум и ставим его на первое место в неотсортированной части массива. И так продолжаем до тех пор, пока неотсортированная часть массива не уменьшится до одного элемента.

```
template<class T>
void selectSort(T a[], int size) {
  for (int i = 0; i < size; i++) {
    int k = i;
    // Find minimum element
    for (int j = i + 1; j < size; j++)
      if (a[j] < a[k])
        k = j; // - current minimum index
    swap(a[k], a[i]);  // Swap current and minimum elements
  }
}
```

Сортировка вставками
На каждом шаге алгоритма мы берем один из элементов массива, находим позицию для вставки в уже отсортированной части и вставляем. Стоит отметить что массив из 1-го элемента считается отсортированным. Нумерация элементов массива начинается с 0 и заканчивается n-1. Основной цикл алгоритма начинается не с 0-го элемента а с 1-го, потому что элемент до 1-го элемента будет нашей отсортированной последовательностью (помним что массив состоящий из одного элемента является отсортированным) и уже относительно этого элемента с номером 0 мы будем вставлять все остальные. Собственно код:

```
for(int i = 1;i < n; i++)
    for(int j = i; j > 0 && x[j - 1] > x[j]; j--)
    swap(x[j - 1], x[j]);
```

Функция swap меняет местами элементы x[j-1] и x[j]. Вложенный цикл ищет место для вставки.
Сортировка обменами (пузырьком)
Обходим массив от начала до конца, попутно меняя местами неотсортированные соседние элементы. В результате первого прохода на последнее место «всплывёт» максимальный элемент. Теперь снова обходим неотсортированную часть массива (от первого элемента до предпоследнего) и меняем по пути неотсортированных соседей. Второй по величине элемент окажется на предпоследнем месте. Продолжая в том же духе, будем обходить всё уменьшающуюся неотсортированную часть массива, запихивая найденные максимумы в конец.

```
for (int i = 0; i < N; i++)
    for (int j = 0; j < N - i; j++)
        if (x[j] > x[j + 1])
            swap(x[j], x[j + 1])
```

Быстрая сортировка *Алгоритм:*

1. Выбираем опорный элемент
2. Разбиваем массив на 3 части
   • Создаём переменные l и r — индексы соответственно начала и конца рассматриваемого подмассива
   • Увеличиваем l, пока l-й элемент меньше опорного
   • Уменьшаем r, пока r-й элемент больше опорного
   • Если l всё ещё меньше r, то меняем l-й и r-й элементы местами, инкрементируем l и декрементируем r
   • Если l вдруг становится больше r, то прерываем цикл
3. Повторяем рекурсивно, пока не дойдём до массива из 1 элемента

```
void quick_sort(int *x, int size) {
    int i = 0, j = size - 1;
    int mid = x[size / 2];
    do {
        while(x[i] < mid) i++;
        while(x[j] > mid) j--;
        if (i <= j) {
            swap(x[i], x[j]);
            i++;
            j--;
        }
    } while (i <= j);

    if(j > 0)
        quick_sort(x, j + 1);
    if (i < size)
        quick_sort(&x[i], size - i);
}
```

Сложность

Сортировка	Выбором	Вставками	Обменами	Быстрая
Худший случай	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
Лучший случай	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n \log n)$
Средний случай	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n \log n)$

[IT-сообщество, *Статьи про сортировку на Хабре*]

dop 7. Операционные системы. Управление оперативной памятью в вычислительной системе. Алгоритмы и методы организации и управления страничной оперативной памятью.

Операционная система — комплекс программ, используемых для управления ресурсами компьютера и предоставления интерфейса пользователю. В понятие управление ресурсами входит *выделение* ресурсов для программ (например, памяти и процессорного времени), *защита* от доступа программ к ресурсам, которыми они не владеют, а также *абстрагирование* 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 памяти** – выбор стратегии организации ОП (реализация поддержки аппарата виртуальной памяти), защита ОП от несанкционированного доступа, задачи корректности работы процесса с выделенной ему ОП.
- **Планирование** – распределение времени центрального процессора, организация и обработка очередей обмена(решается 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 исполняемая в данный момент программа. 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 предоставляемого 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 откачивания во внешнюю память, чтобы на освободившееся пространство поместить данные другого процесса.

Существует несколько моделей по управлению ОП:

• Одиночное непрерывное распределение – все адресное пространство разделяется на два компонента: 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 пространста ОС, иначе - пространства процесса.

• Распределение неперемещаемыми разделами – все адресное пространство делится на две части: одна часть отводится под ОС, остальная - делится на N частей (разделов) и определяется под работу прикладных процессов. Два варианта распределения процесов: обща очередь и очередь к каждому разделу. Необходимо использовать два регистра границ - регситры, 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 совпали — доступ разрешен.

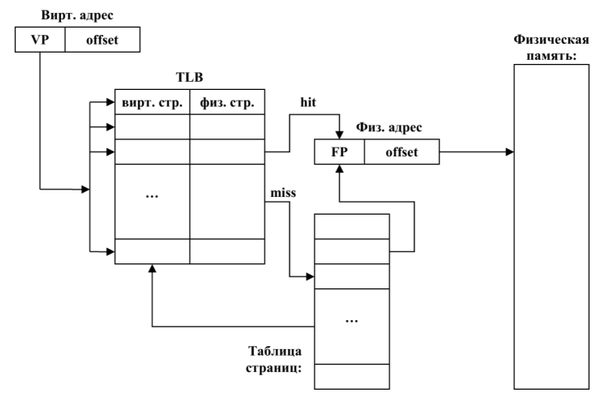
• Распределение перемещаемыми разделами. Исполняемый код процесса может перемещаться по оперативной памяти. Локальная компрессия - система для высвобождения памяти передвигает небольшое количество процессов. Глобальная компрессия - система приостанавливает работу всех процессов и начинает их перемещать, 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.

• Страничное распределение. Все адресное пространство – совокупность блоков фиксированного размера, которые называются **страницами**. **Виртуальное адресное пространство** - пространство с адресами которого оперирует программа. **Физическое адресное пространство** - пространство, которое есть в наличие в компьютере.

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

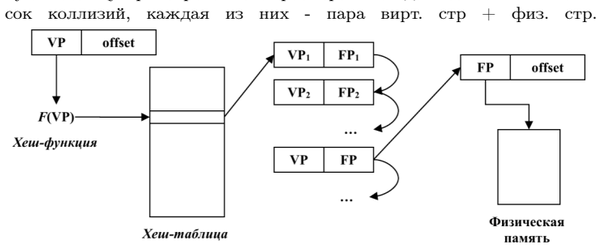
Страничное распределение. Устанавливается соответствие между виртуальными и физическими страницами. Для преобразования виртуального адреса в физический используется таблица страниц. Типовая структура записи таблицы страниц содержит информацию о номере физической страницы, а также совокупность атрибутов,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

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

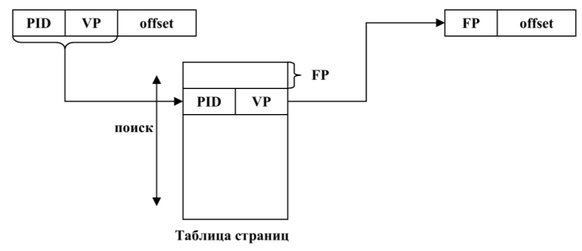


• Иерархическая организация таблиц страниц. Информация о странице представляется в виде совокупности номеров, используя которые, можно получить номер соответствующей физической страницы.

• Хеширование. Извлекается номер виртуальной страницы, который подается в хэш-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 8. Зависимости в реляционных отношениях: функциональные, многозначные, проекции/соединения. Проектирование реляционных БД на основе принципов нормализации отношений. Нормальные формы.

Зависимости в реляционных отношениях

❑ задана переменная отношения R (=таблица), и X и Y являются произвольными подмножествами заголовка R (<составными> атрибутами, наборами столбцов).

Атрибут Y **функционально зависит** от атрибута $X \iff$ каждому значению X соответствует в точности одно значение Y. В этом случае говорят также, что атрибут X **функционально определяет** атрибут Y (X является детерминантом (определителем) для Y, а Y является зависимым от X). (далее, FD (Functional Dependency) — функциональная зависимость).

FD $A \Rightarrow B$ — **тривиальная**, если $A \supseteq B$. Любая тривиальная FD всегда выполнима.

Армстронг предложил правила вывода новых FD на основе существующих. Аксиомы Армстронга:

1. Если $A \supseteq B$, то $A \Rightarrow B$ (**рефлексивность**).
2. Если $A \Rightarrow B$, то $(A\ UNION\ C) \Rightarrow (B\ UNION\ C)$ (**пополнение**).
3. Если $A \Rightarrow B$ и $B \Rightarrow C$, то $A \Rightarrow C$ (**транзитивность**).

В переменной отношения R с атрибутами A, B, C (в общем случае, составными) имеется **многозначная зависимость** B от A ($A \Rightarrow\Rightarrow B$) $\iff$ множество значений атрибута B, соответствующее паре значений атрибутов A и C, зависит от значения A и не зависит от значения C.

Декомпозицией отношения R называется замена R на совокупность отношений $\{R_1, R_2, \dots, R_n\}$ такую, что каждое из них есть проекция R, и каждый атрибут R входит хотя бы в одну из проекций декомпозиции. То есть все R_i состоят только из атрибутов R и любой атрибут R есть хотя бы в одном R_i.

❑ $R' = NATURAL\ JOIN\ (R_1, \dots, R_n)$. Декомпозиция $\{R_1, R_2, \dots, R_n\}$ называется **декомпозицией без потерь**, если $R' = R$

❑ ∃ некоторое отношение r со схемой R, а также два произвольных подмножества атрибутов $A, B \subseteq R$. ❑ $C = R \setminus (A \cup B)$. В этом случае B **многозначно зависит** от A, тогда и только тогда, когда множество значений атрибута B, соответствующее заданной паре $[a : A; c : C]$ отношения r, зависит от a и не зависит от c.

❑ R — переменная отношения, а $A, B, \dots, Z$ — некоторые подмножества множества её атрибутов. Если декомпозиция любого допустимого значения R на отношения, состоящие из множеств атрибутов $A, B, \dots, Z$, является декомпозицией без потерь, говорят, что переменная отношения R удовлетворяет зависимости соединения $*\{A, B, \dots, Z\}$.

Проектирование реляционных БД

Теорема Хита. ❑ Пусть $r\{A, B, C\}$ - отношение, состоящее из множества атрибутов A, B и C. Если в R есть функциональная зависимость $A \Rightarrow B$, то R равно соединению его проекций {A, B} {A, C}.

Нормальные формы

Переменная отношения **находится в 1НФ** тогда и только тогда, когда в любом допустимом значении отношения каждый его кортеж содержит только одно значение для каждого из атрибутов. Реляционное отношение уже находится в 1НФ.

Замыкание множества FD S — множество FD S^+, включающее все FD, логически выводимые из множества S.

FD с минимальным детерминантом (удаление любого атрибута из детерминанта приводит к изменению замыкания S^+ , т. е. порождению множества FD, неэквивалентного S) называется **минимальной слева**.

Атрибут B **минимально зависит** от атрибута A, если выполняется минимальная слева FD $A \Rightarrow B$.

Потенциальный ключ — в реляционной модели данных — подмножество атрибутов отношения, удовлетворяющее требованиям уникальности и минимальности (несократимости).

• Уникальность означает, что нет и не может быть двух кортежей данного отношения, в которых значения этого подмножества атрибутов совпадают (равны).

Свойство уникальности определяется не для конкретного значения переменной отношения в тот или иной момент времени, а по всем возможным значениям, то есть следует из внешнего знания о природе и закономерностях данных, которые могут находиться в переменной отношения.

• Минимальность (несократимость) означает, что в составе потенциального ключа отсутствует меньшее подмножество атрибутов, удовлетворяющее условию уникальности. Иными словами, если из потенциального ключа убрать любой атрибут, он утратит свойство уникальности.

Первичный ключ — в реляционной модели данных один из потенциальных ключей отношения, выбранный в качестве основного ключа (или ключа по умолчанию).

Суперключ отношения r — любое подмножество K заголовка r, включающее, по меньшей мере, хотя бы один возможный ключ r.

FD $A \Rightarrow C$ называется **транзитивной**, если существует такой атрибут B, что имеются ФЗ и $A \Rightarrow B$ и $B \Rightarrow C$ и отсутствует ФЗ $C \Rightarrow A$.

Переменная отношения **находится в 2НФ** тогда и только тогда, когда она находится в 1НФ, и каждый неключевой атрибут минимально функционально зависит от первичного ключа.

Данные находятся в 1 НФ но не 2 НФ:

PK: Модель	PK: Фирма	Цена	Скидка
M5	BMW	50k$	5%
X5M	BMW	51k$	5%
GT-R	Nissan	47k$	10%

Цена машины зависит от модели и фирмы. Скидка зависят от фирмы, то есть зависимость от первичного ключа неполная. Исправляется это путем декомпозиции на два отношения, в которых не ключевые атрибуты зависят от ПК.

PK: Модель	PK: Фирма	Цена
M5	BMW	50k$
X5M	BMW	51k$
GT-R	Nissan	47k$

PK: Фирма	Скидка
BMW	5%
Nissan	10%

Переменная отношения **находится в 3НФ** тогда и только тогда, когда она находится в 2НФ и каждый неключевой атрибут нетранзитивно функционально зависит от первичного ключа.

Данные находятся в 2 НФ но не 3 НФ:

PK: Модель	Магазин	Телефон
BMW	Salon Lan-bin	18-05-00
Audi	Salon Lan-bin	18-05-00
Nissan	Cherep-avto	07-04-99

В отношении атрибут «Модель» является первичным ключом. Личных телефонов у автомобилей нет, и телефон зависит исключительно от магазина. Таким образом, в отношении существуют следующие функциональные зависимости: Модель → Магазин, Магазин → Телефон, Модель → Телефон. Зависимость Модель → Телефон является транзитивной, следовательно, отношение не находится в 3НФ. В результате разделения исходного отношения получаются два отношения, находящиеся в 3НФ:

PK: Модель	Магазин
BMW	Salon Lan-bin
Audi	Salon Lan-bin
Nissan	Cherep-avto

PK: Магазин	Телефон
Salon Lan-bin	18-05-00
Cherep-avto	07-04-99

BCNF, 4НФ Между 3 и 4 нормальной формой есть еще и промежуточная нормальная форма, она называется – **Нормальная форма Бойса-Кодда (BCNF)**. Иногда ее еще называют «Усиленная третья нормальная форма».

1. Таблица должна находиться в третьей нормальной форме. Здесь все как обычно, т.е. как и у всех остальных нормальных форм, первое требование заключается в том, чтобы таблица находилась в предыдущей нормальной форме, в данном случае в третьей нормальной форме
2. Ключевые атрибуты составного ключа не должны зависеть от неключевых атрибутов

Отсюда следует, что требования нормальной формы Бойса-Кодда предъявляются только к таблицам, у которых первичный ключ составной. Таблицы, у которых первичный ключ простой, и они находятся в третьей нормальной форме, автоматически находятся и в нормальной форме Бойса-Кодда.

Требование **четвертой нормальной формы (4NF)** заключается в том, чтобы в таблицах отсутствовали нетривиальные многозначные зависимости и была в 3NF. В таблицах многозначная зависимость выглядит следующим образом: **Таблица должна иметь как минимум три столбца, допустим A, B и C, при этом B и C между собой никак не связаны и не зависят друг от друга, но по отдельности зависят от A, и для каждого значения A есть множество значений B, а также множество значений C.**

В данном случае многозначная зависимость обозначается вот так: $A \Rightarrow B, A \Rightarrow C$ Если подобная многозначная зависимость есть в таблице, то она **не соответствует** четвертой нормальной форме.

[IT-сообщество, *Какие-то ссылки про БД*]

dop 6. Язык ассемблера как машинозависимый язык низкого уровня. Организация ассемблерной программы, секции кода и данных (на примере ассемблера nasm или masm). 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 понятных 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) волен сам определять, какими ресурсами (регистрами и ячейками памяти) воспользоваться для хранения промежуточных результатов, какой алгоритм применить 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 мнемониках) в машинные коды. После трансляции получается объектный файл.
2. *Компоновка (связывание)*. На входе компоновщика один или несколько объектных 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;
fmt:db "a=%d, eax=%d", 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             ; set up stack frame
    mov ebp,esp

    mov eax, [a]         ; put a from store into register
    add eax, 2           ; a+2
    push eax             ; value of a+2
    push dword [a]       ; value of variable a
    push dword fmt       ; address of ctrl string
    call printf          ; Call C function
    add esp, 12          ; pop stack 3 push times 4 bytes

    mov esp, ebp         ; takedown stack frame
    pop ebp              ; same as "leave" op

    mov eax,0            ; normal, no error, return value
    ret                  ; return
```

[*Лекции асм 1 потока*]

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

❑ $\sigma = \langle Const, Func, Pred \rangle$ — некоторая сигнатура, в которой определяются термы и атомы.

«заголовок» ::= «атом»
«тело» ::= «атом» | «тело», «атом»
«правило» ::= «заголовок» ← «тело»;
«факт» ::= «заголовок»;
«утверждение» ::= «правило» | «факт»
«программа» ::= «пусто» | «утверждение» «программа»
«запрос» ::= □ | ? «тело»

❑ $G = ?C_1, C_2, \dots, C_m$ — запрос. Тогда

• атомы $C_1, C_2, \dots, C_m$ называются подцелями запроса G,

• переменные множества называются целевыми переменными,

• запрос □ называется пустым запросом,

• запросы будем также называть целевыми утверждениями.

Полисемантичность – одна и та же логическая программа имеет две равноправные семантики, два смысла.

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

Декларативная семантика	Операционная семантика
Правило $A_0 \leftarrow A_1, A_2, \dots, A_n$;	
Если выполнены условия $A_1, A_2, \dots, A_n$, то справедливо и утверждение A_0.	Чтобы решить задачу A_0, достаточно решить задачи $A_1, A_2, \dots, A_n$.
Факт A_0;	
Утверждение A_0 считается верным.	Задача A_0 объявляется решенной.
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 $C_1, C_2, \dots, C_m$.

С точки зрения декларативной семантики, программные утверждения D и запросы G — это логические формулы, программа P — это множество формул (база знаний), а правильный ответ на запрос — это такие значения переменных (подстановка), при которой запрос оказывается логическим следствием базы знаний.

❑ P – логическая программа, G – запрос к P с множеством целевых переменных $Y_1, Y_2, \dots, Y_k$. Тогда всякая подстановка $\theta = \{Y_1/t_1, \dots, Y_k/t_k\}$ называется ответом на запрос G к программе P. Ответ $\theta = \{Y_1/t_1, \dots, Y_k/t_k\}$ называется **правильным ответом** на запрос G к программе P, если $P \models \forall Z_1 \dots \forall Z_n G\theta$, где $\{Z_1, \dots, Z_n\}$ — множество свободных переменных в термах t_i.

Теорема об основном правильном ответе ❑ $G = ?C_1, C_2, \dots, C_m$ – запрос к хорновской логической программе P. ❑ $Y_1, Y_2, \dots, Y_k$ — целевые переменные, $t_1, t_2, \dots, t_k$ —основные термы. Тогда подстановка $\theta = \{Y_1/t_1, \dots, Y_k/t_k\}$ является правильным ответом на запрос G к программе P тогда и только тогда, когда $P \models (C_1 \wedge \dots \wedge C_m)\theta$.

Под операционной семантикой понимают правила построения вычислений программы. Компьютеру важно «знать», как проводить вычисление программы. Такое понимание программы называется **операционной** семантикой программы. Результат работы логической программы — это правильный ответ на запрос к программе. Значит, операционная семантика должна описывать метод вычисления правильных ответов.

Подстановка θ – **унификатор** выражений E_1, E_2, если $E_1\theta = E_2\theta$.

Подстановка θ – **наиболее общий унификатор (НОУ)** выражений E_1, E_2, если θ – унификатор выражений E_1, E_2 и для любого унификатора η существует такая подстановка ρ, для которой верно $\eta = \theta\rho$.

Пусть

• $G = ?C_1, C_2, \dots, C_m$ — целевое утверждение, в котором выделена подцель C_i,

• $D = A_0 \leftarrow A_1, A_2, \dots, A_n$ —вариант некоторого программного утверждения, в котором $Var_G \cap Var_D = 0$,

• $\theta \in$ НОУ(C_i, A_0) – **наибольший общий унификатор** подцели C_i и заголовка программного утверждения A_0.

Тогда запрос $P = ?(C_1, \dots, C_{i-1}, A_1, A_2, \dots, A_n, C_{i+1}, \dots, C_{i+1}, \dots, C_m)$ - SLD-резольвентой программного утверждения D и запроса G с выделенной подцелью C_i и унификатором θ.

❑ $G = ?C_1, C_2, \dots, C_m$ - целевое утверждение, $P = \{D_1, D_2, \dots, D_m\}$ - хорновская логическая программа. Тогда частичным SLD-резолютивным вычислением, порожденным запросом G_0 к логической программе P называется последовательность троек (конечная или бесконечная)
$(D_{j_1}, \theta_1, G_1), (D_{j_2}, \theta_2, G_2), \dots, (D_n, \theta_n, G_n), \dots$ где

• $D_{ji} \in P, \theta_i \in Subst, G_i$ — целевое утверждение (запрос);

• запрос G_i - SLD-резольвента программного утверждения D_{j_i} и запроса G_{i-1} с унификатором θ_i.

Частичное SLD-резолютивное вычисление
$comp = (D_{j_1}, \theta_1, G_1), (D_{j_2}, \theta_2, G_2), \dots, (D_n, \theta_n, G_n)$ называется **успешным вычислением**, если $G_n = \square$, **бесконечным вычислением**, если comp - это бесконечная последовательность, **тупиковым вычислением**, если comp - это конечная последовательность, при этом для запроса G_n невозможно построить ни одной SLD-резольвенты.

❑ $G = ?C_1, C_2, \dots, C_m$ - целевое утверждение с целевыми переменными $Y_1, Y_2, \dots, Y_k$, $P = \{D_1, D_2, \dots, D_m\}$ — хорновская логическая программа, $comp = (D_{j_1}, \theta_1, G_1), (D_{j_2}, \theta_2, G_2), \dots, (D_n, \theta_n, G_n)$ - успешное SLD-резолютивное вычисление, порожденное запросом G к программе P. Тогда подстановка $\theta = \{\theta_1, \theta_2, \dots, \theta_n\}$, представляющая собой композицию всех вычисленных унификаторов $\theta_1, \theta_2, \dots, \theta_n$, ограниченную целевыми переменными $Y_1, Y_2, \dots, Y_k$ называется вычисленным ответом на запрос G_0 к программе P. У нас есть два типа ответов:

• **правильные ответы**, которые логически следуют из программы;

• **вычисленные ответы**, которые конструируются по ходу SLD-резолютивных вычислений.

Теорема корректности операционной семантики относительно декларативной семантики ❑ $G = ?C_1, C_2, \dots, C_m$ - целевое утверждение, $P = \{D_1, D_2, \dots, D_m\}$ - хорновская логическая программа. θ - вычисленный ответ на вопрос G к программе P. Тогда θ является правильным ответом на вопрос G к программе P.

Теорема полноты ❑ θ - правильный ответ на вопрос G к хорновской логической программе P. Тогда существует такой вычислительный ответ η на запрос G к программе P, что θ=η*p для некоторой подстановки p.

Отображение R, которое сопоставляет каждому непустому запросу G = ? C_1,C_2,...,C_m одну и ту же из подцелей C_i = R(G) в этом запросе, называется **правилом выбора подцелей**.

Для заданного правила выбора подцелей R вычисление запроса G к логической программе называется **R-вычислением**, если на каждом шаге вычисления очередная подцель в запросе выбирается по правилу R. Ответ, полученный в результате успешного R-вычисления, называется **R-вычисленным**.

Теорема сильной полноты Каково бы ни было правило выбора подцелей R, если θ - правильный ответ на запрос G к хорновской логической программе P, то существует такой R-вычисленный ответ η, что равенство $\theta = \eta$p.

Деревом SLD-резолютивных вычислений запроса G к логической программе P называется помеченное корневое дерево $T_{G,P}$, удовлетворяющее следующим требованиям:

• Корнем дерева является исходный запрос G.

• Потомками аждой вершины G являются всевозможные SLD-резольвенты запроса G.

• Листовыми вершинами вляются пустые запросы и запросы, не имеющие SLD-резольвент.

Стратегией вычисления запросов к логическим программам называется алгоритм построения (обхода) дерева SLD-резолютивных вычислений всякого запроса G к произвольной логической программе P. Стратегия вычислений называется **вычислительно полной** , если для любого запроса G и любой логической программы P эта стратегия строит (обнаруживает) все успешные вычисления запроса G к программы P.

Стратегия обхода в ширину: дерево строится (обходится) поярусно — вершина i-го яруса не строится, до тех пор пока не будут построены все вершины $(i-1)$-го яруса. Эта стратегия является полной, но вычислительно затратной.

Стратегия обхода в глубину с возвратом: ветви дерева обходятся поочередно (слева направо) — очередная ветвь дерева не обходится, до тех пор пока не будут пройдены все вершины текущей ветви. Не является полной, но быстрая. Она и является стандартной стратегией выполнения логических программ.

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

dop 2. Графы, деревья, планарные графы; их свойства. Оценка числа деревьев.

Определения

• **Графом** называется произвольное множество элементов V и произвольное семейство E пар из V. *Обозначение*: $G = (V, E)$. Если элементы из E рассматривать как неупорядоченные пары, то граф называется **неориентированным**, а пары называются **рёбрами**. Если же элементы из E рассматривать как упорядоченные, то граф **ориентированный**, а пары — **дуги**.

Пара вида (a, a) называется **петлёй**.

Если пара (a, b) встречается в семействе E несколько раз, то она называется **кратным ребром** (*кратной дугой*).

В дальнейшем условимся граф без петель и кратных рёбер называть простым графом; граф без петель, в котором допустимы кратне ребра — *мультиграфом*; граф, в котором допустимы петли, с петлями — *псевдографом*. В дальнейшем, если не оговаривается обратное, под графом будем подразумевать простой граф.

• Две вершины графа называются **смежными**, если они соединены ребром. Говорят, что вершина и ребро **инцидентны**, если ребро содержит вершину. **Степенью вершины** ($deg\ v$) называется количество рёбер, инцидентных данной вершине. Для псевдографа полагают учитывать петлю дважды.

• **Деревом** называется связный граф без циклов.

• **Остовное дерево** графа – это дерево, подграф данного графа, с тем же числом вершин, что и у исходного графа. Неформально говоря, остовное дерево получается из исходного графа удалением максимального числа рёбер, входящих в циклы, но без нарушения связности графа. Остовное дерево включает в себя все n вершин исходного графа и содержит $n-1$ ребро.

• Дерево, в котором одна из вершин выделена, называется **корневым деревом**. Выделенная вершина называется корнем.

• ❑ имеются корневые деревья $D_i = (V_i, E_i)$, $i = \overline{1,m}$ с корнями v_i, и V_i попарно не пересекаются. Тогда граф $D = (V, E)$, полученный следующим образом: $V = \bigvee_{i=1}^{m} V_i \cup \{v\}$, где $v \notin V_i$, $i = \overline{1,m}$, $E = \bigvee_{i=1}^{m} E_i \cup \{(v, v_1), \dots, (v, v_m)\}$, и в котором выделена вершина v, называется **корневым деревом**.

• **Упорядоченным корневым деревом** называется корневое дерево, в котором задан порядок поддеревьев и каждое поддерево является упорядоченным поддеревом.

• **Планарный граф** — граф, который можно изобразить на плоскости без пересечений рёбер. Если имеется планарная реализация графа и мы <разрежем> плоскость по всем линиям этой реализации, то плоскость распадётся на **грани** (одна из граней бесконечна, она называется внешней гранью).

• Два графа G_1 и G_2 называются **изоморфными**, если существует биекция $f : V_1 \to V_2$ такая, что любые вершины v и w графа G_1 смежны $\Leftrightarrow$ $f(v)$ и $f(w)$ смежны в графе G_2.

• Говорят, что граф G' получен из графа G **подразбиением ребра** $e = (v, w) \in E$, если $V' = V \cup \{u\}$, где $u \notin V$; $E' = E \setminus \{(v, w)\} \cup \{(v, u), (u, w)\}$. Граф G' называется **подразбиением графа** G, если G' может быть получен из G конечным числом подразбиений ребер.

• Графы G_1 и G_2 называются **гомеоморфными**, если найдутся изоморфные их подразбиения G_1' и G_2' соответственно.

Свойства

• **Утв 1.** ❑ в графе $G = (V, E)$ p вершин и q ребер ($|V| = p, |E| = q$) и $degV_i$ - степень вершин V_i, тогда $\sum_{i=1}^{p} degV_i = 2q$.

▲ Равенство следует из того, что слева каждое ребро учитывается два раза. ■

• **Лемма 1.** Если граф $G = (V, E)$ связный и ребро (a, b) содержится в некотором цикле в графе G, то при выбрасывании из графа G ребра (a, b) снова получится связный граф.

▲ Если путь из V_i в V_j не проходил через (a, b), то все останется как есть. Если выкинуть (a, b) и (a, b) принадлежал циклу, то все останется как есть. ■

• **Теорема 1.** Любой связный граф содержит хотя бы одно остовное дерево.

▲ ❑ $G = (V, E)$ - связный граф. Если в G нет циклов, то G -искомое остовное дерево. Если в G ∃ цикл, то выкинем из G ∀ ребро (a, b) из цикла, тогда по Лемме 1 останется связный подграф. Если в нем нет циклов, то он - искомое остовное дерево. Иначе пролжаем процесс выбрасывания ребер из циклов. Процесс конечный, так как граф конечный. ■

• **Лемма 2.** Если к связному графу добавить новое ребро на тех же вершинах, то появится цикл.

▲ ❑ $(V_i, V_j) \notin E, i \neq j$. Так как граф связный, то в нем ∃ путь из V_i в $V_j \implies$ ∃ простая цепь из V_i в $V_j \implies$ эта цепь + ребро (V_j, V_i) - цикл в данном подграфе ■

• **Лемма 3.** ❑ в графе $G = (V, E)$ p вершин и q рёбер. Тогда 1) в G не менее $p - q$ связных компонент. 2) Если при этом в G нет циклов, то G состоит ровно из $p - q$ связных компонент.

▲ 1) 👁 граф $G_0 = (V, \emptyset)$ и будем из G_0 получать граф $G = (V, E)$, добавляя по одному ребру из E. В G_0 p связных компонент. При добавлении одного ребра к любому графу (на тех же вершинах) число связных компонент либо не уменьшается, либо уменьшается на 1. $\implies$ при добавлении к графу q ребер число св. компонент может уменьшиться на $\leq q \implies$ в графе G будет $\geq$ p-q св. компонент. (т.к. в G_0 p св. компонент)

2) Если в графе G нет циклов, то каждое добавленное ребро должно соединять две вершины из разных св. компонент (иначе по лемме 2 появляется цикл) $\implies$ число св. компонент уменьшается ровно на 1 $\implies$ в G ровно p-q св. компонент. ■

• **Теорема об эквивалентных определениях дерева.** ❑ $G = (V, E)$, $|V| = p$, $|E| = q$, тогда следующие утверждения попарно эквивалентны:

1. G — дерево.
2. G — граф без циклов и $q = p - 1$.
3. G — связный граф и $q = p - 1$.
4. G — связный граф, но при удалении любого ребра становится несвязным.
5. G — граф без циклов, но при добавлении любого ребра в нём образуется цикл.

▲ 1 → 2 *Дано*: G - связный, без циклов.*Док-тъ*: G - без циклов и q = p-1.
По лемме 3 s=p-q=1

2 → 3 *Дано*: G - без циклов, q = p-1.*Док-тъ*: G - связный и q = p-1.
По лемме 3 s = p-q =1 $\implies$ G - связный

3 → 4 *Дано*: G - связный, q = p-1.*Док-тъ*: G - связный, но при удалении любого ребра становится несвязным.
Из G удаляем ∀ ребро - получаем G'. Для $G's' \geq p - q' = 2$. По лемме 3 G' - несвязен

5 → 5 *Дано*:G - связный, но при удалении любого ребра становится несвязным.*Док-тъ*: G - без циклов, но при добавлении любого ребра в нём образуется цикл.
❑ в G есть цикл l и l - ребро из G, G' = G-l. По лемме 1 - связный (Противоречие)

5 → 1 *Дано*:G - без циклов, но при добавлении любого ребра в нём образуется цикл.*Док-тъ*: G - связный, без циклов.
Если в G найдется цикл, то удалим из G любое ребро из цикла. Останется связный граф - противоречие. Значит, G без циклов ■

Теорема: Оценка числа деревьев. Число упорядоченных корневых деревьев с q рёбрами не превосходит 4^q.

▲ ❑ $(D; v_0)$ — упорядоченное корневое дерево с q ребрами. Обойдем дерево D в глубину из вершины $v_0 \in V$ по порядку его поддеревьев. При таком обходе по каждому ребру пройдем два раза: первый раз при переходе в соответствующее поддерево, второй раз при возвращении из него. По этому обходу построим код дерева D — набор k(D) из нулей и единиц длины 2q. Сначала этот код не заполнен. При проходе по очередному ребру заполняем в коде k(D) первый незаполненный разряд по следующим правилам:
1) если по ребру переходим в поддерево, то в код k(D) пишем ноль;
2) если по ребру возвращаемся из поддерева, то в код k(D) пишем единицу. Тогда различным упорядоченным корневым деревьям соответствуют разные коды. Поэтому $\delta''(q)$ не превосходит числа наборов из нулей и единиц длины 2q, т.е. $\delta''(q) \leq 2^{2q} = 4^q$. ■

Теорема: формула Эйлера. Для любой планарной реализации связного планарного графа $G = (V, E)$ с p вершинами, q ребрами и r гранями выполнено $p - q + r = 2$.

Теорема Понтрягина-Куратовского. Для любой реализации планарного графа $G = (V, E)$ планарен тогда и только тогда, когда в нем не найдется ни одного подграфа, гомеоморфного K_5 или $K_{3,3}$.

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

dop 9. 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.

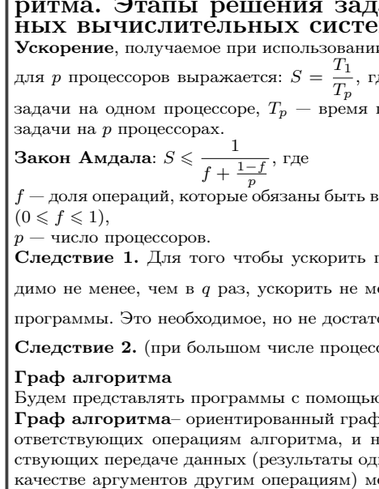
Закон Амдала: $S \leqslant \frac{1}{f + \frac{1-f}{p}}$, где

f — доля операций, которые обязаны быть выполнены последовательно ($0 \leqslant f \leqslant 1$),
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 необходимое, 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 графом 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.
Особенностями 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)



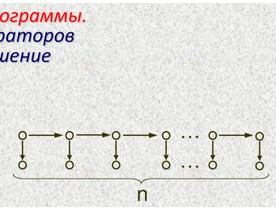
Выделяют два типа отношений: операционное и информационное.
Операционное отношение: две вершины A и B соединяются направленной дугой $A \Rightarrow B$ тогда и только тогда, когда вершина B может быть выполнена сразу после вершины A.
Информационное отношение: две вершины A и B соединяются направленной дугой $A \Rightarrow B$ тогда и только тогда, когда вершина B использует в качестве аргумента некоторое значение, полученное в вершине A.

Операционный граф: вершины — операции, дуги — операционные отношения.
Граф операционной истории программы: вершины — срабатывания операторов, дуги — операционные отношения.
Информационный граф: 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** графа.

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];
}
```

(1) (2)



Критический путь графа алгоритма — это длина самого длинного пути от <входа> графа к его <выходу> (от первого оператора к 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
Ширина ЯПФ — это максимальная ширина ярусов в ЯПФ.
Высота ярусно-параллельной формы — это сложность параллельной реализации алгоритма/программы. ЯПФ определяется неоднозначно.
Ярусно-параллельная форма называется **канонической**, если у любой вершины, кроме вершин 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 чем 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 11. Классификация языков, определяемых конечными автоматами, регулярными выражениями и праволинейными грамматиками. Эквивалентность и минимизация конечных автоматов.

Грамматики и регулярные множества и выражения

- **Грамматика** — это четверка $G = (N, T, P, S)$, где N — **алфавит нетерминальных символов**, T — **алфавит терминальных символов**, $N \cap T = \varnothing$, P — **конечное множество правил** вида $\alpha \to \beta$, 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 $\gamma \to \delta \in P$, то $\alpha\gamma\beta \Rightarrow \alpha\delta\beta$, $\forall\alpha,\ \beta \in (N \cup T)^*$.

- **Сентенциальная форма** грамматики G — цепочка, выводимая из ее начального символа.

- **Языком**, порождаемым грамматикой G ($L(G)$) называется множество всех её терминальных сентенциальных форм $L(G) = \{w | w \in T^*, S \Rightarrow^*_G w\}$.

- Грамматики **эквивалентны**, если они порождают один и тот же язык.

- **Классификация грамматик по Хомскому:**

Тип 0 Любая порождающая грамматика.
Тип 1 (Неукорачивающая или контекстно-зависимая грамматика) Если каждое правило кроме $S \to e$ имеет вид $\alpha \to \beta$, где $|\alpha| \leqslant |\beta|$, и в том случае, когда $S \to e \in P$, S не встречается в правых частях правил.
Тип 2 (Контекстно-свободная грамматика.) Если каждое правило имеет вид $A \to \beta$, где $A \in N$, $\beta \in (N \cup T)^*$.
Тип 3 (Регулярная праволинейная (или леволинейная) грамматика.) Каждое правило имеет вид либо $A \to xB$, либо $A \to x$, где $A,\ B \in N,\ x \in T^*$. (для леволинейных поменять xB на Bx.)

- **Типом языка** называется максимальный тип грамматики, порождающей этот язык.

- **Регулярное множество** определяется рекурсивно:

1. $\varnothing$ — 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^* (итерация).
5. Ничто другое не является регулярным множеством.

- **Регулярное выражение** определяется рекурсивно:

1. $\varnothing$ — регулярное выражение, обозначающее рег. множество $\varnothing$.
2. e — регулярное выражение, определяющее рег. множество $\{e\}$.
3. a — регулярное выражение, определяющее рег. множество $\{a\}$.
4. Если p и q — регулярные выражения, то регулярными являются $(p|q)$ (обозначает множество $P \cup Q$), pq (обозначает множество PQ), p^* (обозначает множество P^*).
5. Ничто другое не является регулярным выражением.

Конечные автоматы
Для распознавания регулярных множеств служат конечные автоматы.

- **Недетерминированный КА** — пятёрка $M = (Q, T, D, q_0, F)$ где Q — **конечное множество состояний**, T — **входной алфавит**, D — **функция переходов**, отображающая множество $Q \times (T \cup \{e\})$ во множество подмножеств Q, $q_0 \in Q$ — **начальное состояние**, $F \subseteq Q$ — **множество заключительных состояний**.

- **Детерминированный КА** — НКА, в котором $D(q, e) = \varnothing$ для любого $q \in Q$ и $D(q, a)$ содержит не более 1 элемента для любых $q \in Q$ и $a \in T$.

- **Теорема 1.** Язык, допускаемый ДКА, — множество всех его допустимых цепочек — является регулярным множеством.

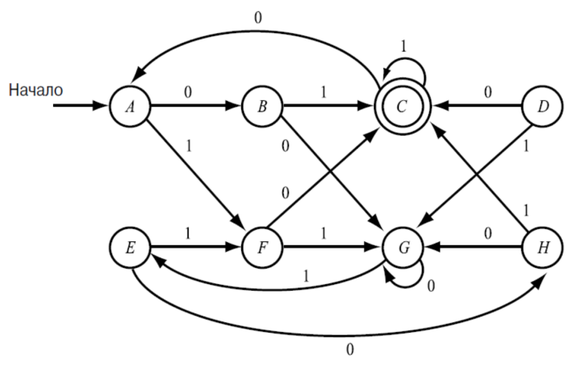
- **Теорема 2.** Пусть $G = (N, T, P, S)$ — праволинейная. Тогда существует НКА $M = (Q, T, D, q_0, F)$, для которого $L(M) = L(G)$.

- **Теорема 3.** Для каждого НКА M существует праволинейная грамматика G такая, что $L(M) = L(G)$. По НКА можно построить эквивалентный ему ДКА. По ДКА можно построить эквивалентный ДКА с минимальным числом состояний.

Состояние s конечного автомата M эквивалентно состоянию t конечного автомата N тогда и только тогда, когда автомат M, начав работу в состоянии s будет допускать в точности те же цепочки, что и автомат N, начавший работу в состоянии t.
Два автомата M_1 и M_2 называются эквивалентными, если их входные алфавиты совпадают и для каждого состояния M_1 существует эквивалентное ему состояние M_2.
Алгоритм минимизации:

1. Удаляем все недостижимые состояния и те, из которых нет пути в конечное состояние.
2. Разбиваем исходное множество состояний на два, первое из которых состоит из всех конечных состояний, а второе — из всех остальных: $\{F\},\ \{Q \setminus F\}$.
3. Если элементы одного множества при переходе по одному символу попадают в разные множества, то разбиваем исходное множество так, чтобы в одном множестве оказались только те состояния, которые переходят в состояния одного множества. Например пусть есть множества $\{s_1, s_2, s_3, s_4\}, \{s_5, s_6\}$ и переходы $s_1 \xrightarrow{a} s_1$, $s_2 \xrightarrow{a} s_1$, $s_3 \xrightarrow{a} s_5$, $s_4 \xrightarrow{a} s_6$. Тогда новые множества будут иметь следующий вид: $\{s_1, s_2\}$, $\{s_3, s_4\}$, $\{s_5, s_6\}$.
4. Продолжаем разбиение до тех пор, пока это возможно.
5. Каждое множество состояний старого автомата заменяем на одно состояние нового.

➤ **Пример**



	A	B	C	D	E	F	G	H
A								
B	x							
C	x	x						
D	x	x	x					
E	0	x	x	x				
F	x	x	x	0	x			
G	x	x	x	x	x	x		
H	x	0	x	x	x	x	x	

Там где крестик - состояния неэквивалентны. Перебираем все возможные переходы из каждого состояния. В данном случае из каждого состояния по 0 и 1. Если два состояния переходят по одному символу в разные состояния, то ставится крестик (они уже неэквивалентны)

[Игнатьев, *Лекция по формалкам 4 слайды*, page 25-37]

dop 13. Жизненный цикл программного обеспечения (ПО). Основные виды деятельности при разработке ПО. Каскадная и итерационная модели жизненного цикла.

Весь период существования ПО, связанный с подготовкой к его разработке, разработкой, использованием и модификациями, до того момента, когда полностью прекращается всякое ее использование, называют **жизненным циклом ПО**.
Процессы жизненного цикла ПО строятся из отдельных видов деятельности (activities). Стандартом ISO/IEC 12207 Standard for Information Technology определены 74 вида деятельности, связанной с разработкой и поддержкой ПО. Основные из них (думаю, из этого знать нужно только выделенное, как основные типы деятельности):

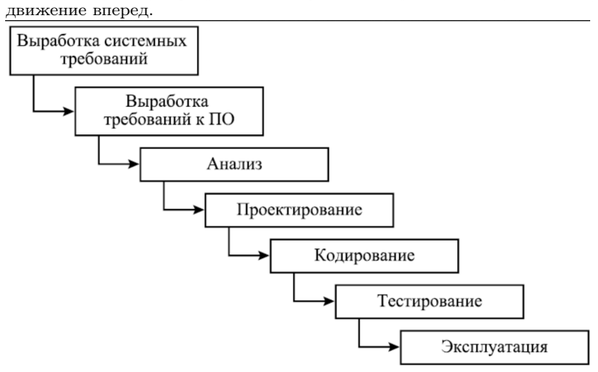
- **Приобретение ПО**: инициация приобретения, подготовка запроса предложений, подготовка контракта, анализ поставщиков, получение ПО и завершние приобретения.

- **Разработка ПО**: развертывание процесса разработки, анализ системных требований, проектирование (программно-аппаратной) системы в целом, анализ требований к ПО, проектирование архитектуры ПО, детальное проектирование, кодирование и отладочное тестирование, интеграцию ПО, квалификационное тестирование ПО, системную интеграцию, квалификационное тестирование системы, развертывание (установку или инсталляцию) ПО, поддержку процесса получения ПО.

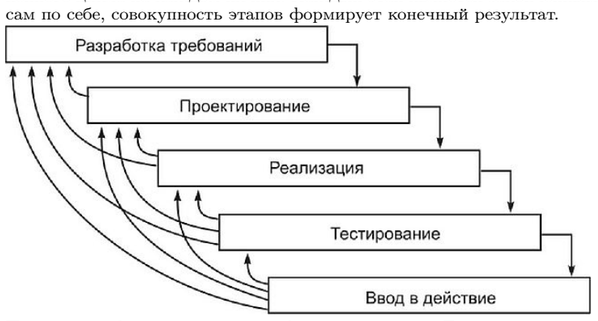
- **Поддержка ПО**: развертывание процесса поддержки, анализ возникающих проблем и необходимых изменений, внесение изменений, экспертизу и передачу измененного ПО, перенос ПО с одной платформы на другую, изъятие ПО из эксплуатации.

- **Управление проектом**: запуск проекта и определение его рамок, планирование, выполнение проекта и надзор за его выполнением, экспертизу и оценку проекта, свертывание проекта.

Каскадная и итерационная модели жизненного цикла ПО
Каскадная модель. Основная суть **каскадной** модели в том, что этапы зависят друг от друга и следующий начинается, когда закончен предыдущий, образуя таким образом поступательное (каскадное) движение вперед.



Параллелизм этапов в каскадной модели, хоть и ограничен, но возможен для абсолютно независимых между собой работ. При этом интеграция параллельных кусков все равно происходит на каком-то следующем этапе, а не в рамках одного.
Команды разных этапов между собой не коммуницируют, каждая команда отвечает четко за свой этап.
Недостатками этой модели являются получение результата по прохождению всех этапов и сложность выявления ошибок. Возвращаться назад трудно. Не понятно что возвращать: если произошел сбой на каком-то этапе, его последствия видны только в конце.
Для заказчиков данная модель выглядит линейно и со стороны достаточно просто: из завершенного этапа проектирования следует программирование, а затем тестирование - и так шаг за шагом пока не будет достигнута финальная точка и цель, ради которой ведется разработка. Данная модель понятно и чисто укладывается в документы, например в договора и роадмапы при наличии четко обозначенных контрольных точек. В любой момент времени можно легко понять была ли пройдена та или иная точка контроля или нет, и соблюдены ли сроки. По этим причинам долговременные и особо крупные проекты, рассчитанные на десятилетия и вовлечение большого числа организаций-участников, руководствуются преимущественно каскадной моделью.
Однако представление о простоте каскадной модели является иллюзорным. Оно появляется из-за ограниченного видения клиентом всего процесса, ведь данная модель не подразумевает вовлечение заказчика в детали процессов разработки, и демонстрирует понятный и конечный результат работы только на контрольных точках и в конце проекта.
В реальности каскадную модель нельзя назвать простой, на практике ею сложно управлять. Внесение заказчиком значительных изменений в процессе разработки по каскадной модели или срабатывание серьезных не предусмотренных проектом рисков несут разрушительный характер для всего процесса - модель приходиться перестраивать, графики перепланировать.
Итерационная модель. Итерационная модель предполагает разбиение проекта на части (этапы, итерации) и прохождение этапов жизненного цикла на каждом из них. Каждый этап является законченным сам по себе, совокупность этапов формирует конечный результат.



Поясним разбиение на этапы на бытовом примере. Допустим, нам нужен стол в гостиную.

- На первом этапе мы сделает столешницу, ножки и скрепим их так, чтобы стол стоял.

- На втором, укрепим и покрасим его.

- А на третьем, накроем скатертью и купим подходящие к нему стулья. На каждой итерации мы работали с одним и тем же продуктом и в конце каждой итерации получали результат, которым можно пользоваться (естественно, с определенными ограничениями).

С каждым этапом разработка приближается к конечному желаемому результату или уточняются требования к результату по ходу разработки, и соответственно в любой момент текущая итерация может оказаться последней или очередной на пути к завершению.
Данный подход позволяет бороться с неопределенностью, снимая ее этап за этапом, и проверять правильность технического, маркетингового или любого другого решения на ранних стадиях.
Использование итерационной модели снижает риски глобального провала и растраты всего бюджета, получение несинхронизированных ожиданий и ошибочного понимания процессов как клиентом, так и каждым участником команды разработки. Оно также дает возможность завершения разработки в конце любой итерации (в каскадной модели вы должны прежде завершить все этапы).

[Evergreen, *Какая-то ссылка про подходы к разработке*]

dop 15. Основные понятия криптографии. Односторонняя функция с секретом. Протокол Диффи-Хеллмана выработкиобщего 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.
Шифротекст (криптограмма) — результат шифрования открытого текста.
Шифр — семейство обратимых отображений множества последовательностей открытых текстов во множество последовательностей шифротекстов.
Ключ — параметр, определяющий выбор конкретного отображения.
Зашифрование — процесс преобразования открытого текста в шифрованный 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: $X \to Y$, обладающая свойствами:

1. $\exists$ полиномиальный алгоритм вычисления значений $f(x)$.
2. $\nexists$ полиномиального алгоритма обращения функции f.

Односторонняя функция с секретом (с лазейкой) — функция $f_k(x) : X \to Y$ зависящая от параметра k, называемым секретным ключом или лазейкой и такая, что:

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. Почти для всех k и $y \in Y$, нахождение $f_k^{-1}(y)$ вычислительно неосуществимо без знания k.

Один из примеров, претендующих на то, чтобы являться односторонней функцией с лазейкой — функция $f(x) = x^m (\mathrm{mod}\ n)$, 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 mod n, требует примарного разложения n [типо разложение на простые числа]. Эта информация может считаться лазейкой.
Простые поля Галуа – поля классов вычетов по модулю простого числа. Пример: $Z_3 = \{n(mod 3)\} = \{0, 1, 2\}$
Порождающие элементы мультипликативной группы поля [м.г. – типо все элементы поля без нуля] называют его **примитивными элементами**.
Пример: числа 2, 6, 7, 8 – примитивные элементы поля $\mathbb{F}_{11}$
Проверяем 2:

k	1	2	3	4	5	6	7	8	9	10
$2^k(mod 11)$	2	4	8	5	10	9	7	3	6	1

$ord(2) = 10 \implies$ 2 – примитивный. Проверим, например, 3:

k	1	2	3	4	5
$3^k(mod 11)$	3	9	5	4	1

$ord(3) = 5 \neq 10 \implies$ 3 – не примитивный.
Как ускорить проверку примитивности:
$p = 11, p - 1 = 2 \cdot 5$, проверяем степени $\frac{10}{2} = 5$ и $\frac{10}{5} = 2$ элементов 2 и 3:
$2^2(mod 11) = 4 \neq 1, 2^5(mod 11) = 10 \neq 1 \implies$ 2 – примитивный.
$3^2(mod 11) = 9 \neq 1, 3^5(mod 11) = 1 \implies$ 3 – не примитивный.
Выработка общего секретного ключа по открытому каналу связи (протокол Диффи-Хеллмана)
Алиса (А) и Боб (В) обмениваются сообщениями по открытому каналу.
Для обеспечения секретности нужен общий секретный ключ.

1. А и В выбирают простое число p (число p 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, $X = \alpha^x (\mathrm{mod}\ p)$, $Y = \alpha^y (\mathrm{mod}\ p)$. X и Y передаются друг другу по открытому каналу.
4. А вычисляет: $K = Y^x (\mathrm{mod}\ p)$. В вычисляет: $K = X^y (\mathrm{mod}\ p)$.
5. Таким образом, А и В получают общий секретный ключ $K = \alpha^{xy} (\mathrm{mod}\ p)$, который в дальнейшем используется в алгоритмах симметричного шифрования.

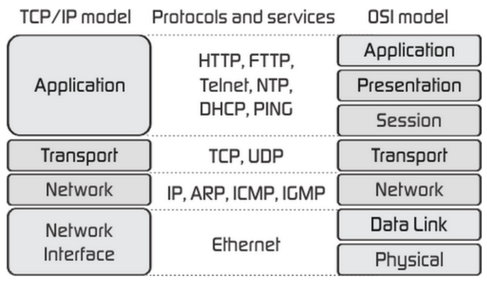
Пассивный злоумышленник, перехвативший, но не изменяющий сообщений не может определить ключ K: его определение связано с решением одного из ур-ей шага 3, а это вычислительно трудная задача дискретного логарифмирования.
Протокол Диффи-Хеллмана подвержен атаке **"человек по середине"**: если к каналу связи имеет доступ активный злоумышленник, то выработав два ключа – общий с А и общий с В, он может представляться Алисе Бобом, а Бобу – Алисой.

[С. И. Гуров, *Запись лекций по курсу прикладная алгебра. Алгебраические основы кодирования и шифрования*, page 139-151]

dop 16. 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 (дейтаграммы) 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) вышележащего уровня в протокольную единицу нижележащего уровня. PDU вышележащего уровня не доступны на нижележащем уровне (нижележащий уровень не понимает структуры данных вышележащего уровня).
5. Каждый уровень имеет строго определенный интерфейс с нижележащим и вышележащим уровнями и набор сервисов, которые может использовать вышележащий 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 вышерасположенного 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 — проблемы синтаксиса и семантики передаваемой информации.
Приложений – обеспечение работы часто используемых приложений, например — передача файлов.

TCP/IP:
Прикладной уровень. На прикладном уровне (Application layer) работает 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 транспортного 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 путём 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, определяющих 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 приёмнику и являются основой протокола маршрутизации. Для формулирования алгоритмов маршрутизации сеть рассматривается как граф. При этом маршрутизаторы являются узлами, а физические линии между маршрутизаторами — рёбрами 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 $C_8 = (C_1, C_2, ..., C_7)$ - вектор расстояния до R_8
- Изначально $C = (\infty, \infty, ..., \infty)$

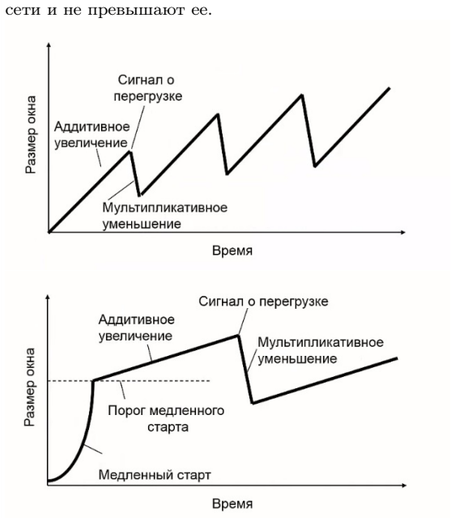
1. Каждые T секунд R_i шлёт C_i всем свои соседям
2. Если R_i нашёл более дешёвый путь, то он обновляет C_i у всех своих соседей
3. Вернуться к 1

- Длину вектора C устанавливает администратор

Алгоритм Дейкстры (пример алгоритма по состоянию канала):

1. Определение топологии сети: каждый маршрутизатор передаёт лавиной всем своим соседям состояния своих линий и строит топлогию сети (1. Периодически, 2 Когда изменяется состояние линии)
2. Вычисление по алгоритму Дейкстры: каждый маршрутизатор независимо запускает алгоритм Дейкстры наикратчайшего 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 разнообразных 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.



[Смелянский, *Компьютерные сети, том 1*] [Смелянский, *Компьютерные сети, том 2*]

dop 14. Качество программного обеспечения и методы его контроля. Тестирование и другие методы верификации.

Основные определения

- **Качество ПО** в стандарте ISO 9126 — вся совокупность его характеристик, относящихся к возможности удовлетворять высказанные или подразумеваемые потребности всех заинтересованных лиц.
- Внутреннее качество связано с характеристиками ПО самого по себе, без учета его поведения; внешнее качество характеризует ПО с точки зрения его поведения; качество ПО при использовании в различных контекстах — качество, которое ощущается пользователями при конкретных сценариях работы ПО.
- **Верификация** означает проверку того, что ПО разработано в соответствии со всеми требованиями к нему, или что результаты очередного этапа разработки соответствуют ограничениям, сформулированным на предшествующих этапах.
- Валидация — проверка того, что сам продукт правилен, то есть подтверждение того, что он действительно удовлетворяет потребностям и ожиданиям пользователей, заказчиков и других заинтересованных сторон.
- **Методы обеспечения качества** представляют собой техники, гарантирующие достижение определенных показателей качества при их применении.

Методы верификации

- Методы и техники выяснения свойств ПО во время его работы. Это, прежде всего, все виды тестирования.
- Методы и Техники определения качества на основе симуляции работы ПО с помощью моделей: проверка на моделях (model checking), прототипирование (макетирование для оценки качества принимаемых решений).
- Методы и Техники выявления нарушений формализованных правил построения исходного кода ПО, проектных моделей и документации: инспектирование кода.
- Методы и Техники обычного или формализованного анализа проектной документации и исходного кода для выявления их свойств: методы анализа архитектуры ПО.

Виды тестирования

- Стрессовое (нагрузочное) тестирование — проверяет производительность ПО в условиях повышенных нагрузок.
- Тестирование черного ящика (тестирование на соответствие) — проверка требований спецификаций, стандартов, ограничений.
- Функциональное тестирование — его частный случай, проверка требований к функциональности.
- Аттестационное тестирование — для получения документа о соответствии.
- Тестирование белого ящика — на основе знаний о структуре системы.
- Тестирование на отказ — попытка вывести систему из строя в том числе некорректными данными.
- Тестирование с помощью набора мутантов — программ, отличных от тестируемой в отдельных точках.
- Модульное тестирование — проверка отдельных модулей. Важная часть отладочного тестирования. **Предусловия** описывают для каждой операции, на каких входных данных она предназначена работать, **постусловия** — как должны соотноситься входные данные с возвращаемыми ею результатами, **инварианты** — определяют критерии целостности внутренних данных модуля.
- Интеграционное тестирование — проверка правильности взаимодействия некоторого набора модулей друг с другом.
- Системное тестирование — проверка правильности работы системы в целом, ее способности правильно решать поставленные пользователями задачи в различных ситуациях.
- Тестирование пользовательского интерфейса — имитация действий пользователей над элементами этого интерфейса. Частные случаи — тестирование графического интерфейса, тестирование интерфейса Web-приложений.

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

dop 12. Функции FIRST и FOLLOW. LL(l)-грамматики. Конструирование таблицы предсказывающего анализатора.

LL(1)-анализатор – нисходящий алгоритм синтаксического разбора. Цифра 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 анализатора 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(α) как множество терминалов, с которых начинаются строки, выводимые из α. Если $\alpha \Rightarrow^* e$, то e также принадлежит FIRST (α).
- Определим FOLLOW(A) для нетерминала A как множество терминалов a, которые могут появиться непосредственно справа от A в некоторой сентенциальной 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 сентенциальной формы, то e также принадлежит FOLLOW (A).
- Грамматика $G = (N, T, P, S)$ является $LL(1)$-*грамматикой* тогда и только тогда, когда для каждой пары правил $A \to \alpha$, $A \to \beta$ из P (то есть правил с одинаковой левой частью) выполняются следующие 2 условия:

1. FIRST(α) $\cap$ FIRST(β) $= \varnothing$.
2. Если $e \in$ FIRST(α), то FIRST(β) $\cap$ FOLLOW(A) $= \varnothing$.

Вычисление FIRST для символов КС грамматики
Вход: КС грамматика $G = (N, T, P, S)$.
Выход: Множество FIRST(X) для каждого символа $X \in (N \cup T)$.

1. Если X — терминал, то положить FIRST(X) $= \{X\}$, если нетерминал — FIRST(X) $= \varnothing$.
2. Если в P есть правило $X \to e$, то добавить в FIRST(X) e.
3. Выполнять алгоритм на рисунке ниже.

   Краткое описание: для каждого правила $X \leftarrow Y_1 Y_2 \ldots$ в FIRST(X) добавлять FIRST(Y_1) и, если $e \in$ FIRST(Y_1), добавлять FIRST(Y_2) и, если $e \in \ldots$.

```
do { continue = false;
  Для каждого нетерминала X
        Для каждого правила X -> Y1Y2...Yk
        {i=1; nonstop = true;
        while (i <= k && nonstop)
         {добавить FIRST(Yi) п {e} к FIRST(X);
         if (Были добавлены новые элементы)
                continue = true;
         if (e != FIRST (Yi)) nonstop = false;
         else i+ = 1;
         }
    if (nonstop) {добавить e к FIRST(X);
                continue = true;
  } } }
while (continue);
```

Вычисление FIRST для цепочки
Вход: КС грамматика $G = (N, T, P, S)$.
Выход: Множество FIRST($X_1, X_2, \ldots, X_n$), $X_i \in (N \cup T)$.

1. Для всех $X \in (N \cup T)$ вычислить FIRST(X).
2. Положить FIRST($X_1, X_2, \ldots, X_n$) $= \varnothing$.
3. Выполнить алгоритм на рисунке ниже
   Краткое описание: добавляем во множество FIRST(X_1), и, если $e \in$ FIRST(X_1), FIRST(X_2), и, если $e \in \ldots$.

```
{i = 1; nonstop = true;
   while (i <= && nonstop)
    {добавить FIRST(Xi) п {e} к FIRST(u);
      if (e not in FIRST(Xi) nonstop = false;
        else i+ = 1;
    }
if (nonstop) {добавить e к FIRST(u);
    } }
```

Вычисление FOLLOW для нетерминалов грамматики
Вход: КС грамматика $G = (N, T, P, S)$.
Выход: Множество FOLLOW(X) для каждого символа $X \in N$.

1. FOLLOW(X) $= \varnothing$ для каждого знака $X \in N$.
2. Добавить \$ к FOLLOW(S).
3. Если в P есть правило вывода $A \to \alpha B \beta, \alpha,\ \beta \in (N \cup T)^*$, то все элементы из FIRST(β), кроме e, добавляем к FOLLOW(B).
4. если в P есть правило $A \to \alpha B$ или $A \to \alpha B \beta,\ \alpha,\ \beta \in (N \cup T)^*$, где FIRST($\beta$) содержит e, то есть $\beta \Rightarrow^* = e$, то все элементы из FOLLOW (A) добавить к FOLLOW (B).
5. Продолжать шаги 2 и 3, пока какие-то множества меняются.

Конструирование таблицы предсказывающего анализатора по грамматике G. Пусть $A \to \alpha$ — правило вывода грамматики и $a \in$ FIRST(α). Тогда анализатор делает развертку A по α, если входным символом является a. Трудность возникает, когда $\alpha = e$ или $\alpha \Rightarrow e$. В этом случае нужно развернуть A в α, если текущий входной символ принадлежит FOLLOW(A) или если достигнут \$ и \$ $\in$ FOLLOW(A).
Алгоритм
Вход: КС-грамматика $G = (N, T, P, S)$.
Выход: Таблица $M[A, a]$ предсказывающего анализатора, $A \in N,\ a \in T \cup \{\$\}$.
Для каждого правила вывода $A \to \alpha$ грамматики выполнить шаги 1 и 2. После этого выполнить шаг 3.

1. Для каждого терминала a из FIRST(α) добавить $A \to \alpha$ к $M[A, a]$.
2. Если $e \in$ FIRST(α), добавить $A \to \alpha$ к $M[A, b]$ для каждого терминала b из FOLLOW (A). Кроме того, если $e \in$ FIRST(α) и \$ $\in$ FOLLOW(A), добавить $A \to \alpha$ к $M[A, \$]$.
3. Положить все неопределенные входы равными <ошибка>.

Алгоритм построения таблиц может иметь более одного правила в одной клетке

1. Неоднозначные грамматики
2. Леворекурсивные грамматики

Построена таблица, в которой каждая строка соотносится с некоторым состоянием, каждый столбез – с терминалом, а в ячейках содержатся формулы преобразований. Данная таблица используется для определения следующего состояния.
Грамматики, для которых таблица предсказывающего анализатора не имеет неоднозначно определённых ячеек, называются LL(1)-грамматика:

1. Сканирование слева-направо
2. Левое порождение
3. Предпросмотр 1 символа

Язык называют LL(1)-языком, если для него существует LL(1)-грамматика
Для произвольной LL(1)-грамматики G алгоритм строит таблицу предсказывающего анализатора, распознающего все цепочки из L(G) и только их.
Если G — LL(1)-грамматика, то L(G) — детерминированный КС-язык.

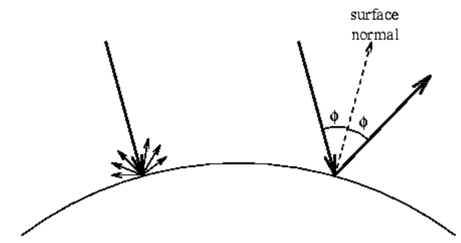
[Пупкин-Залупкин, *Напиши источник!*, page 69-96]

dop 10. Глобальные и локальные модели освещения в компьютерной графике. Модель Фонга.

Глобальные и локальные модели
Моделировать освещение очень важно, т.к. на основе освещения мы воспринимаем форму объектов. Глаз воспринимает освещение и «реконструирует» трехмерную форму. Моделирование освещения — ключевой элемент фотореализма.
Полный отраженный свет = сумма вклада от источников света и других поверхностей сцены.
Светоиспускающие источники — первичные, светоотражающие источники — вторичные. **Локальные модели** учитывают только первичные источники света, **глобальные** — все источники.
Поверхность, не освещаемая источником света, все равно может быть видима
Локальные модели при вычислении освещения в данной точке учитывают только положение этой точки относительно первичных источников света.
Существующие модели:

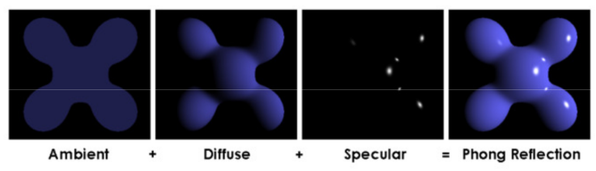
- Диффузное отражение (матовый пластик, дерево, бумага) — модель Ламберта.
- Идеально зеркальное отражение (зеркало) — модель отражения.
- Зеркальное отражение (блики на объекте) — модели Фонга и Блинна.

Диффузное и зеркальное отражение



Модель Фонга
Для определения яркости пикселя можно воспользоваться формулой Y=0.3*R+0.59*G+0.11*B.
Расчёт освещения по Фонгу требует вычисления цветовой интенсивности трёх компонент освещения: фоновой (ambient), рассеянной (diffuse) и глянцевых бликов (specular).
Фоновое освещение — это постоянная в каждой точке величина надбавки к освещению. Вычисляется фоновая составляющая освещения как: $I_a = aL_a$, где a — свойство материала воспринимать фоновое освещение (коэффициент фонового освещения), L_a — мощность фонового освещения.
Рассеянный свет при попадании на поверхность рассеивается равномерно во все стороны. При расчете такого освещения учитывается только ориентация поверхности (нормаль) и направление на источник света. Рассеянная составляющая рассчитывается по закону косинусов (закон Ламберта): $I_d = d\langle S, n\rangle$. Здесь d — свойство материала воспринимать рассеянное освещение (коэффициент рассеянного освещения), S — направление из точки объекта на источник света, n — вектор нормали в точке объекта.
Зеркальный свет при попадании на поверхность подчиняется следующему закону: <Падающий и отраженный лучи лежат в одной плоскости с нормалью к отражающей поверхности в точке падения, и эта нормаль делит угол между лучами на две равные части>. Таким образом отраженная составляющая освещенности в точке зависит от того, насколько близки направления на наблюдателя и отраженного луча: $I_s = k_s\langle r, v\rangle^p$. Здесь k_s — коэффициент зеркального отражения, r — направление отраженного луча, v — направление на наблюдателя, p — коэффициент блеска, свойство материала.
Итого формула освещения в модели Фонга: $I = I_a + I_d + I_s = aL_a + d\langle S, n\rangle + k_s\langle r, v\rangle^p$.

Модель Фонга: пример



[*Презентация по освещению от ИИТа*, page 19-26]

dop 17. Теоретические основы передачи данных, физический уровень стека протоколов. Системы передачи данных Ethernet и Wi-Fi: алгоритмы работы, управление множественным доступом к каналу

Теоретические основы

Данные — описание фактов, явлений.

Сигнал — представление данных при передаче.

Передача — процесс взаимодействия передатчика и приёмника, с целью передачи сигнала. Данные и сигналы могут быть аналоговым (непрерывными) и цифровыми. Для цифровой передачи данных сигнал нужно разбивать на уровни (для кодирования).

Полоса пропускания канала — спектр частот, которые канал пропускает без существенного понижения мощности сигнала.

Скорость передачи зависит от способа кодирования данных на физическом уровне и **сигнальной скорости** – скорости изменения значения сигнала.

Пропускная способность канала — максимальная скорость, с которой канал способен передавать данные.

Теорема Найквиста-Котельникова: $R_{max\_data\_rate} = 2 * D * log_2 L$ bps (bits per second)

- $R_{max\_data\_rate}$ - максимальная пропускная спсобность канала
- D - ширина полосы пропускания канала (максимальная частота сигнала в спектре).
- L - количество уровней (значений) сигнала.

Теорема Котельникова — Аналоговый сигнал $u(t)$, не содержащий частот выше F_{max}(Гц), полностью определяется последовательностью своих значений в моменты времени, отстоящие друг от друга на $\frac{1}{2F_{max}}$

Шум в канале измеряется, как соотношение мощности полезного сигнала к мощности шума: $\frac{S}{N}$ (измеряется в децибелах $1dB = 10 * \log_{10} \frac{S}{N}$)

Теорема Шеннона: $R_{max} = D * log_2(1 + \frac{S}{N})$ bps (bits per second)

- R_{max} - максимальная скорость передачи данных по каналу с шумом
- D - ширина полосы пропускания канала (максимальная частота сигнала в спектре).
- $\frac{S}{N}$ - соотношение сигнал-шум в канале, S - мощность сигнала, N - мощность шума

Характеристики физической среды:

- Полоса пропускания
- Пропускная способность
- Задержка
- Затухание
- Помехоустойчивость
- Достоверность передачи
- Стоимость
- Простота прокладки
- Сложность обслуживания

Ethernet

- Узлы в сети Ethernet адресуются при помощи 6-байтового двоичного числа, называемого MAC-адресом (Media Access Control — управление доступом к носителю).
- Распределением MAC-адресов между производителями оборудования занимается международная некоммерческая организация IEEE (Institute of Elecrical and Electronics Engineers — Институт инженеров электротехники и электроники).
- Любой участник может послать в сеть сообщение, но только тогда, когда в ней <тихо> — нет другой передачи.
- Ethernet использует протокол разрешения адресов (ARP) для определения MAC-адресов и их сопоставления с известными адресами сетевого уровня.
- У каждого узла в IP-сети есть MAC и IP-адреса.
- Узел должен использовать собственные MAC- и IP-адреса в полях источника, а также предоставить MAC и IP-адреса для назначения.
- Несмотря на то, что IP-адрес назначения будет предоставлен более высоким уровнем OSI, отправляющему узлу необходимо найти MAC-адрес назначения для данного канала Ethernet. В этом заключается назначение протокола ARP.

Wi-Fi

- Обычно схема сети Wi-Fi содержит не менее одной точки доступа и не менее одного клиента.
- Также возможно подключение двух клиентов в режиме точка-точка (Ad-hoc), когда точка доступа не используется, а клиенты соединяются посредством сетевых адаптеров <напрямую>.
- Точка доступа передаёт свой идентификатор сети (SSID – Service Set Identifier) с помощью специальных сигнальных пакетов
- Зная SSID сети, клиент может выяснить, возможно ли подключение к данной точке доступа.
- При попадании в зону действия двух точек доступа с идентичными SSID приёмник может выбирать между ними на основании данных об уровне сигнала.
- Стандарт Wi-Fi даёт клиенту полную свободу при выборе критериев для соединения.

Управление множественным доступом

- Проблема управления множественным доступом встаёт в тот момент, когда несколько отправителей хотят отправить свои данные через сеть.
- Протокол множественного доступа может определять и предотвращать коллизии пакетов (кадров) данных при условии, что в качестве режима конкурирующего доступа используется метод доступа к каналу, или зарезервированы ресурсы для установления логического канала.
- Механизм множественного доступа основан на схеме мультиплексирования (передача нескольких потоков данных с меньшей скоростью по одному каналу связи) физического уровня.

Протоколы:

- **CSMA/CD (Carrier Sense Multiple Access with Collision Detection)** Если во время передачи кадра рабочая станция обнаруживает другой сигнал, занимающий передающую среду, она останавливает передачу, посылает сигнал преднамеренной помехи и ждёт в течение случайного промежутка времени (backoff delay), перед тем как снова отправить кадр. В Ethernet коллизии могут быть обнаружены сравнением передаваемой и получаемой информации. Если она различается, то другая передача накладывается на текущую (возникла коллизия) и передача прерывается немедленно. Посылается сигнал преднамеренной помехи, что вызывает задержку передачи всех передатчиков на произвольный интервал времени, снижая вероятность коллизии во время повторной попытки. Ethernet является классическим примером протокола CSMA/CD.
- **Aloha (чистая)** Отправитель начинает передавать, если в этот момент он слышит чужой сигнал, то он понимает, что произошло наложение и перестает передавать, ждет случайное время и начинает передавать сначала. В Wi-Fi суть та же.
- **Aloha (слотированная)** Модификация алохи. Все время работы канала разделяется на слоты. Размер слота при этом должен быть равен максимальному времени кадра. Такая организация работы канала требует синхронизации: одна из станций испускает сигнал начала очередного слота, поскольку передачу теперь можно начинать не в любой момент, а только по специальному сигналу, то время на обнаружение коллизии сокращается вдвое. Размер слота при этом должен быть равен максимальному времени кадра.

[Смелянский, *Компьютерные сети, том 2*, page 69-96]

dop 19. Понятие о парадигме программирования. Основные парадигмы программирования. Языки и парадигмы программирования.

Парадигма программирования — совокупность идей и понятий, определяющих стиль программирования.

Императивная парадигма (процедурная) основана на фоннеймановской модели компьютера. Основные понятия императивных языков программирования представляют собой абстракции основных понятий фон-неймановской модели.

Любой императивный язык программирования включает в себя:

- Понятие переменной — абстрагирует понятие ячейки памяти.
- Понятие операции — обобщает арифметико-логические команды.
- Понятие оператора — абстрагирует общее понятие команды.

Операторы: присваивания, управления (циклы, операторы выбора, перехода и т.д.).

Далее во всех парадигмах рассмотрена задача перевернуть последовательность символов.

C

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAX_ELEMENTS 1024
char * pInput= NULL;
char Buffer[MAX_ELEMENTS];
int main() {
    int current, count = 0;
    pInput= (char*)calloc(1, sizeof(char));
    while (fgets(Buffer, sizeofBuffer, stdin)) {
        int oldCount= count;
        current = strlen(Buffer);
        count += current;
        pInput = (char*)realloc(pInput, count+1);
        if (!pInput) {
            fprintf(stderr, "No memory");
            return 1;
        }  else strcat(pInput+ oldCount, Buffer);
    }
    for (char * pBegin= pInput, * pEnd= pInput + count - 1;
        pBegin< pEnd; ++pBegin, --pEnd) {
        char t = *pBegin;
        *pBegin= *pEnd; *pEnd= t;
    }
    puts(pInput);
    free(pInput);
    return 0;
}
```

Объектная парадигма основана на понятии объекта. Объект обладает состоянием и поведением.

Поведение — взаимодействие с другими объектами посредством сообщений. Для каждого вида сообщения существуют обработчики, которые могут модифицировать состояние и посылать сообщения. Объекты с одинаковым поведением и набором состояний объединяются в классы. Объектная парадигма сочетается с императивной.

Состояние описывается набором переменных, а обработчики сообщений представляют собой процедуры или функции, имеющие доступ к состоянию. Посылка сообщения сводится к вызову соответствующего обработчика.

Объектная парадигма основана на императивной.

C++

```
#include <iostream>
#include <string>

class Circle {
   double radius;
   std::string color;
public:
   Circle(double r = 1.0, std::string c = "red") :
        radius(r), color(c){ }
   double getRadius() { return radius; }
   std::string getColor() { return color; }
   double getArea() { return radius*radius*3.1416; }
};

int main() {
   Circle c1(1.2, "blue");
   std::cout << "Radius=" << c1.getRadius()
        << " Area=" << c1.getArea()
        << " Color=" << c1.getColor() << std::endl;

   Circle c2(3.4);
   std::cout << "Radius=" << c2.getRadius()
        << " Area=" << c2.getArea()
        << " Color=" << c2.getColor() << std::endl;
   return 0;
}
```

Функциональная парадигма основана на понятиях функция и выражение.

Основная операция — вызов функции.

Выражение — это комбинация вызовов функций.

Функции – объекты первого порядка, то есть могут быть значениями переменных, возвращаемыми и передаваемыми значениями, могут быть созданы динамически.

При вызове функции вначале вычисляются выражения — фактические параметры — затем их значения подставляются вместо аргументов в выражение-тело функции. Наконец, вычисляется значение тела, которое и будет значением вызова.

Основные понятия:

- Лисп-выражение — это атом, либо список.
- Атом — это либо символ (идентификатор), либо число.
- Список — это последовательность членов списка, разделенных пробелами, заключенная в круглые скобки.

Член списка — это либо атом, либо список.

- Есть специальный атом — `nil`, который представляет собой пустой список. Это единственный атом, который одновременно является и списком.

Lisp

```
(defun shift (l r)
    ( if (null l)
        r
        (shift (cdrl) (cons (car l) r))
    )
)
(defun reverse1 (s) (shift s nil))
```

Логическая парадигма: парадигма программирования, основанная на математической логике — программы в ней задаются в форме логических утверждений и правил вывода.

Идея – описать семантику задачи в терминах формул исчисления предикатов.

Программа на Прологе = описание предикатов.

Пример: `append(L1, L2, L3)` – истина, если `L3` – конкатенация списков `L1` и `L2`.

Предикаты описываются двояко: факты и правила.

Факт: ИмяПредиката(список_константных_аргументов).

Правило: `P(X1,X2,X3,...) :- P1(X1,...), P2(X1,...,Y,...), ...`

:- означает импликацию, запятая в правой части – логическое "и".

Правила соединяются в единую формулу путем логической операции "или". Фактически получаем единую формулу – дизъюнкцию импликаций (каждое правило – хорновский дизъюнкт).

Prolog

```
reverse1([], []).
reverse1([X|Q], Z) :- reverse1(Q, Y), append(Y, [X], Z).
```

[Головин, *Материалы по языкам программирования к госэкзамену*]

dop 21. 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 единым компьютером.

Свойства децентрализованных 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 краху алгоритма;
4. отсутствие общего источника глобального времени.

пункты 1–3 о недопустимости хранения всей информации необходимой для принятия решения в одном место.

Синхронизация времени

○ **Аппаратные часы** — счетчик временных сигналов и регистр с начальным значением счетчика [из слайдов].

(*Аппаратные часы* — счетчик времени, система содержащая автономный источник питания и регистр. [из вики]).

○ Отношение "произошло до" ($\rightarrow$): $a \rightarrow b$ означает, что все процессы согласны, что сначала 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 $\rightarrow$ транзитивно ($a \rightarrow b$ и $b \rightarrow c$ 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*.

○ ***Логические часы*** — согласованное (между ЭВМ РС) время (потенциально) не имеющее ничего общего с астрономическим временем.

Введем 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 выполнял 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 большими чем 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 "ОК", 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 посылается процессу, следующему за ним, и т.д., пока не найдется работающий процесс.
   - 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 кольцу 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. Децентрализованный алгоритм на основе временных меток.
   Необх. глобальное 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 запрашивал разрешение на вход в нее, то он посылает отправителю сообщение"ОК".
   - Если получатель находится внутри критической секции, то он не отвечает, а запоминает запрос.
   - Если получатель выдал запрос на вхождение 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 посылает сообщение "ОК". Если у чужого запроса метка больше, то ответ не посылается, а чужой запрос запоминается.

   Выход из критической секции:
   - После выхода из секции он посылает сообщение «OK» всем процессам, запросы от которых он запомнил, а затем стирает все запомненные 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*, КонспектЛекций.zip/3-4-MPI-Sync.doc: page 8–15]

dop 23. 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 РС базируются на использовании среды 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 сервис 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 кэшей семантика 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 открывшего файл 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.

Изменения 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, заканчивающуюся 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 проблемы 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 — помещать 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. a) 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). b) кэширование в ядре. (Накладные расходы на обращение к ядру). c) кэш-менеджер в виде отдельного процесса. (Ядро освобождается 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 характера приложений и данных о быстродействии процессоров, памятей, 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 отложенной 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*]

dop 24. 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 <- op y z; x <- op y; x <- y; x[i] <- y; x <- y[i];`,
   - переходы: `goto L; ifTrue x goto L; ifFalse x goto L;`
   - дополнительное: `param x; call x n; return y;`
2. таблицу символов,
   - переменные, их имена в программе и атрибуты, такие как область видимости, тип, для имен функций - число параметров, и т. п.

• LIR (низкий 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),

• Поток управления покидает базовый блок без останова или ветвления, кроме, возможно, последней инструкции 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.

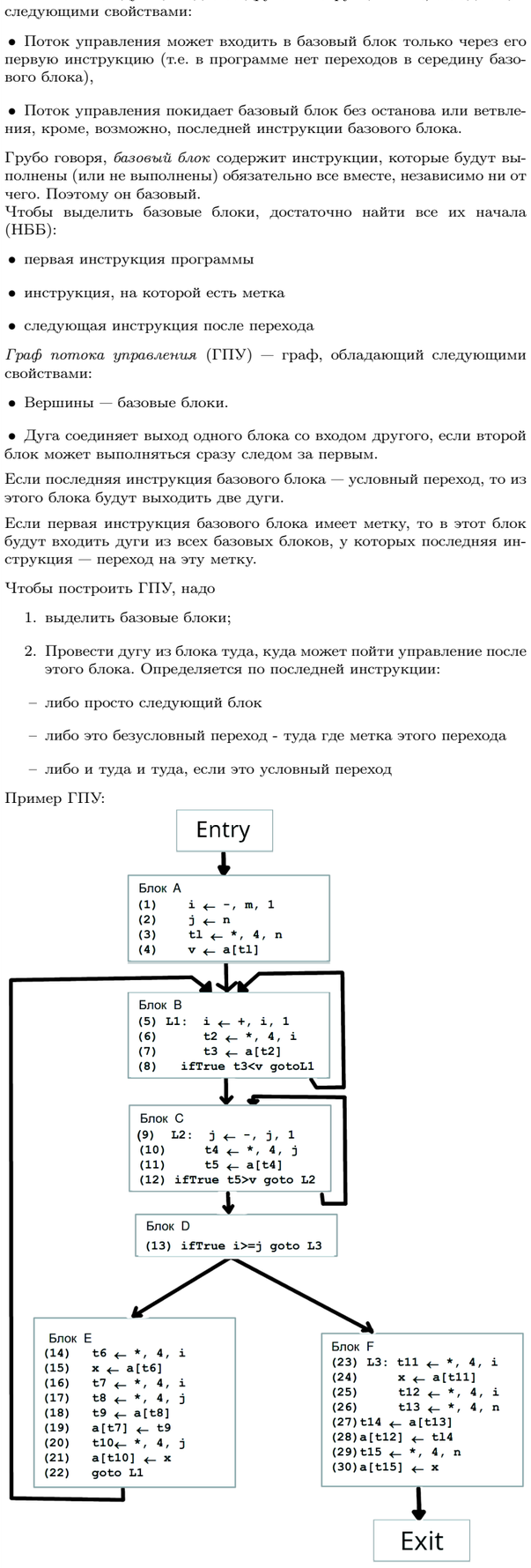
Если последняя инструкция базового блока — условный переход, то из этого блока будут выходить две дуги.

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. Провести дугу 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 22. Отказоустойчивость в распределенных системах. Типы отказов. Фиксация контрольных точек и восстановление после отказа. Репликация и протоколы голосования. Надежная групповая рассылка.

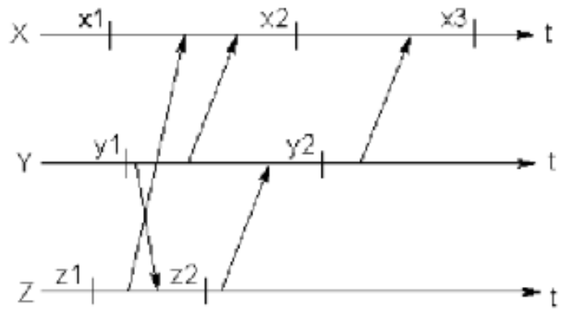
Основные определения

• **Отказом системы** называется поведение системы, не удовлетворяющее ее спецификациям. Отказы могут быть случайными, периодическими или постоянными. Случайные отказы (сбои) при повторении операции исчезают. Отказы по характеру своего проявления: <византийские> (система активна, но некорректно работает), 2) <пропажа признаков жизни> (частичная или полная).

Два подхода - восстановление решения после отказа системы (или ее компонента) и предотвращение отказа системы (отказоустойчивость).

Восстановление:

1. Прямое — основано на своевременном обнаружении сбоя и ликвидации его последствий путем приведения некорректного состояния системы в корректное. Такое восстановление возможно только для определенного набора заранее предусмотренных сбоев.
2. Возвратное — возврат процесса (или системы) из некорректного состояния в некоторое из предшествующих корректных состояний.



На рисунке показаны три процесса (X,Y,Z), взаимодействующие через сообщения. Вертикальные черточки показывают на временной оси моменты запоминания состояния процесса для восстановления в случае отказа. Стрелочки соответствуют сообщениям и показывают моменты их отправления и получения. Предположим теперь, что процесс Z сломается и будет восстановлен в состояние z2. Это приведет к откату процесса Y в y1, а затем и процессов X и Z в начальные состояния x1 и y1. Этот эффект известен как эффект домино.

• Множество контрольных точек называется **строго консистентным**, если во время его фиксации никаких обменов между процессами не было. Оно соответствует понятию строго консистентного глобального состояния, когда все посланные сообщения получены и нет никаких сообщений в каналах связи.

• Множество контрольных точек называется **консистентным**, если для любой зафиксированной операции приема сообщения, соответствующая операция посылки также зафиксирована (нет сообщений-сирот).

Методы фиксации контрольных точек

1. **Простой метод** — фиксация локальной контрольной точки после каждой операции посылки сообщения. При этом посылка сообщения и фиксация должны быть единой неделимой операцией (транзакцией). Множество последних локальных контрольных точек является консистентным (но не строго консистентным).

   Чтобы избежать потерь сообщений при восстановлении с использованием консистентного множества контрольных точек необходимо повторить отправку тех сообщений, квитанции о получении которых стали недействительными в результате отката. Используя временные метки сообщений можно распознавать сообщения-призраки.

2. **Синхронная фиксация.** Два вида контрольных точек - постоянные и пробные.

   Постоянная контрольная точка — это локальная контрольная точка, являющаяся частью консистентной глобальной контрольной точки. *Пробная контрольная точка* — это временная контрольная точка, которая становится постоянной только в случае успешного завершения алгоритма.

   Фиксация: *1 фаза.* Инициатор фиксации (процесс P_i) создает пробную КТ и просит все остальные процессы сделать то же самое. При этом процессу запрещается посылать неслужебные сообщения после того, как он сделает пробную контрольную точку. Каждый процесс извещает P_i о том, сделал ли он пробную КТ. Если все процессы сделали пробные контрольные точки, то P_i превращает пробные точки в постоянные. Если какой-либо процесс не смог сделать пробную точку, все точки отменяются.

   2 фаза. P_i информирует все процессы о своем решении.

   Восстановление: *1 фаза.* Инициатор отката спрашивает остальных, готовы ли они откатываться. Когда все будут готовы к откату, откат.

   2 фаза. P_i сообщает всем о принятом решении. Получив это сообщение, каждый процесс поступает указанным образом. С момента ответа на опрос готовности и до получения принятого решения процессы не должны посылать сообщения.

3. **Асинхронная фиксация.** Множество контрольных точек может быть неконсистентным. При откате происходит поиск подходящего консистентного множества путем поочередного отката каждого процесса в ту точку, в которой зафиксированы все посланные им и полученные другими сообщения.

Репликация - механизм синхронизации содержимого нескольких копий объекта
Рассмотрим возможные протоколы работоспособности коммуникаций и процессоров:

• *Протокол принятия единых решений.*

Все исполнители являются исправными и должны либо все принять, либо все не принять заранее предусмотренное решение.

• *Протокол принятия согласованных решений.*

Наоборот, принятие решения при надежных коммуникациях, но ненадежной работы процессоров. В системе с m неверно работающими процессорами можно достичь согласия только при наличии $2m + 1$ верно работающих процессоров (более 2/3) (задача Византийских генералов).

Предположим, что коммуникации надежны, а процессоры нет.
Алгоритм надежных неделимых широковещательных рассылок сообщений:

Фаза 1 Процесс-отправитель посылает сообщение группе процессов (список их идентификаторов содержится в сообщении). Процессы приписывают сообщению приоритет и помещают в очередь (как недоставленное), информируют отправителя.

Фаза 2 Отправитель получил все ответы => выбирает максимальный полученный приоритет и присваивает его сообщению, рассылает всем процессам. Получатели принимают, помечают сообщение как доставленное, упорядочивают сообщения в своих очередях.

Если получатель обнаружит, что он имеет сообщение с пометкой <недоставленное>, отправитель которого сломался, то он для завершения выполнения протокола осуществляет следующие два шага в качестве координатора: опрашивает всех о статусе сообщения; получив все ответы, реагирует на них. Если сообщение у какого-то получателя помечено как <доставленное>, то его окончательный приоритет рассылается всем. Получив это сообщение каждый процесс выполняет шаги фазы 2. Иначе координатор заново начинает весь протокол с фазы 1.

[*Курс распределенных систем*]

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 объемлющей 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 строчка) — 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 объектные 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 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
Целые 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:

• $(-1)^S * M * 2^p$, где S — бит знака, M — нормализованная мантисса ($0.5 \le M < 1$), 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 C 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 C, 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 ассоциированы 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 индекса (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 C).

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 (АТД) — 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 25. Локальная оптимизация при компиляции программы. Ориентированный ациклический граф и метод нумерации значений.

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),

• Поток управления покидает базовый блок без останова или ветвления, 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

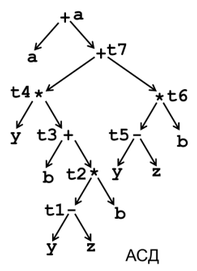
Все указанные преобразования для локальной оптимизации можно выполнить за один просмотр ББ, если представить его в виде ориентированного ациклического графа (ОАГ). Суть ОАГ заключается в том, что узлы, представляющие одинаковые значения, присутствуют в единственном экземпляре.

Пример. Выражение в исходном коде:

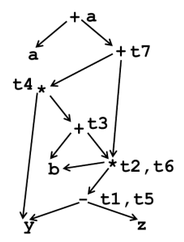
```
a = a + y*(b +(y-z)*b)+(y-z)* b
```

```
t1 ← -, y, z
t2 ← *, t1, b
t3 ← +, b, t2
t4 ← *, y, t3
t5 ← -, y, z
t6 ← *, t5, b
t7 ← +, t4, t6
a  ← +, a, t7
```

Выражение в промежуточном представлении



АСД



ОАГ

ОАГ можно представить в виде таблицы значений (вычислять таблицу проще для понимания). Пример таблицы ниже.
Каждая строка таблицы значений представляет один узел ОАГ. Строка содержит:

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 операндов (у унарных 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 ← Op_i, l_i, r_i`.
Функция `#val(s)` определяет номер значения, определяемого сигнатурой `s = (Op, #val(l), #val(r))` .

```
for each "t_i ← Op_i, l_i, r_i" do
   s_i = (Op_i, #val(l_i), #val(r_i))
   if(ТЗ содержит s_j == s_i)
      then
            вернуть j в качестве значения #val(s_i)
      else
            завести в ТЗ новую строку ТЗ_k
            записать сигнатуру s_i в строку ТЗ_k
            вернуть k в качестве значения #val(s_i)
```

```
t1 ← -, y, z
t2 ← *, t1, b
t3 ← +, b, t2
t4 ← *, y, t3
t5 ← -, y, z
t6 ← *, t5, b
t7 ← +, t4, t6
a  ← +, a, t7
```

(a) Блок E до оптимизации

```
t1^5 ← -, y^3, z^4
t2^6 ← *, t1^5, b^2
t3^7 ← +, b^2, t2^6
t4^8 ← *, y^3, t3^7
t5^5 ← -, y^3, z^4
t6^6 ← *, t5^5, b^2
t7^9 ← +, t4^8,t6^6
a^10 ← +, a^1, t7^9
```

(c) Блок E после нумерации значений

```
t1 ← -, y, z
t2 ← *, t1, b
t3 ← +, b, t2
t4 ← *, y, t3
t5 ← t1
t6 ← t2
t7 ← +, t4, t6
a  ← +, a, t7
```

(c) Блок E 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	4	t1, t5
6	*	5	2	t2, t6
7	+	2	6	t3
8	*	3	7	t4
9	+	8	6	t7
10	+	1	9	a
# значения	КОП	# операнда	# операнда	Присоединенные переменные
	Определение значения (сигнатура)			

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,

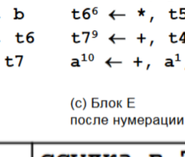
• $Output$ – множество переменных, используемых после выхода из блока B.

Живыми называются переменные, значения которых, вычисленные в рассматриваемом базовом блоке, используются в других базовых блоках.

Пример. Рассмотрим базовый блок $B = \langle P, \{a, b, c, d\}, \{a, b\}\rangle$

```
a ← +, b, c
b ← -, b, d
c ← +, c, d
e ← +, b, c
```

Базовый блок B до удаления мертвого кода



ОАГ базового блока B

```
a ← +, b, c
b ← -, b, d
```

Новые значения `c` и `e` можно не вычислять, так как `c` и `e` не входят в множество *Output*

Базовый блок B после удаления мертвого кода

Восстановление базового блока по его ОАГ

• Для каждого узла с одной или несколькими связанными переменными строится трехадресная инструкция, которая вычисляет значение одной из этих переменных.

• Если у узла несколько присоединенных живых переменных, то следует добавить команды копирования, которые присвоят корректное значение каждой из этих переменных.

[*Презентации Гайсаряна*, slides 29-45]

dop 27. Постановка задачи дискретной оптимизации. Метод ветвей и границ. Задача целочисленного линейного программирования.

Постановка задачи дискретной оптимизации: найти $\min\limits_{x\in X} f(x)$, где X - конечно или счетно (мн-во допуст. значений перем. x), в постановке м.б. и нахождение и max

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 допустимых значений переменной x. В основе МВГ лежит следующая идея: если нижняя граница значений функции 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, нижняя 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 и достигается на соответствующей подобласти.
Основная задача линейного программирования (озЛП) $(c_1, c_2, \dots, c_n)$, найти

$$\max_{x\in R,\ Ax\leqslant b} \langle c, x\rangle.$$

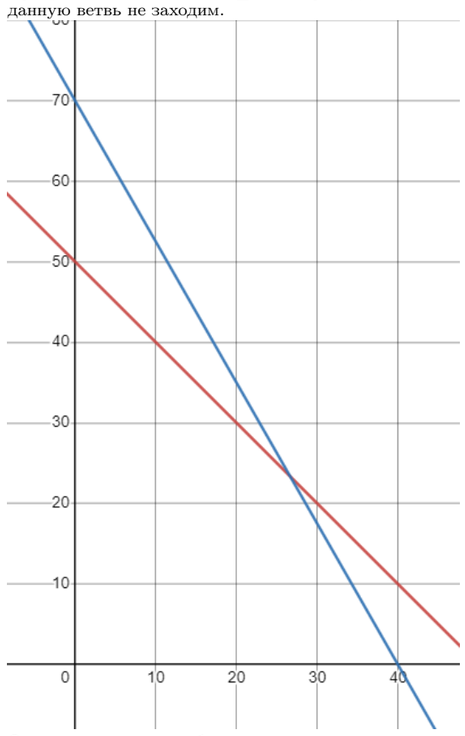
Каноническая задача ЛП:

$$\max_{Ax=b,\ x\geqslant 0} \langle c, x\rangle.$$

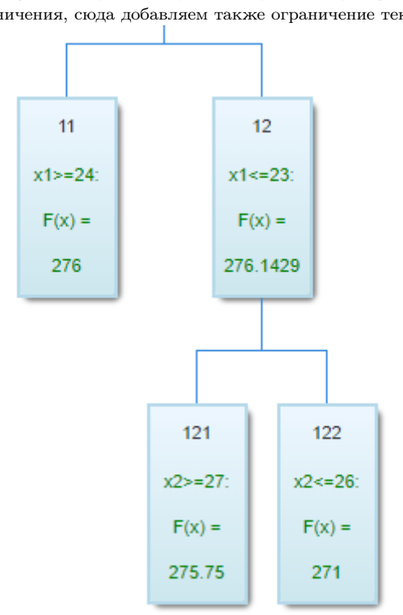
Формально данные задачи не являются дискретными, но они могут быть сведены к перебору конечного числа угловых точек (вершин полиэдра, задающего ограничения) на основании принципа граничных решений:
Если задача имеет решение, то найдется такая подматрица A_I матрицы A, что любое решение системы уравнений $A_I x = b_I$ реализует максимум.
Для задачи целочисленного ЛП на переменные накладывается требование их принадлежности мн-ву целых чисел:

$$\max_{x\in Z,\ Ax\leqslant b} \langle c, x\rangle.$$

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 области ограничения $x_1 \leq -x_2 + 50$ и $x_1 \leq -\frac{7}{4}x_2 + 70$, точка пересечения $x1 = \frac{70}{3} = 23.333, x2 = \frac{80}{3} = 26.667, Z = 5 * \frac{70}{3} + 6 * \frac{80}{3} = 276.667$ – не явл.целочисл. Решение лежит внутри области, заданной ограничениями. Мы проверяем другие конечные точки, проводя линию, задающие целочисл ограничения для x_1, x_2 по области. Выбираем переменную для ветвления (иногда выбор важен, 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, переходим к след. ветви $x_2 \geq 27 \Rightarrow \{x_1 = 22.75, x_2 = 27, Z = 275.75\}$ — нецелочисл. и $\leq$ рекорда, ветвимся теперь по x_1: строим $x_1 \leq 23 \Rightarrow \{x_1 = 23, x_2 = 26.857, Z = 276.142\} >$ рекорда, теперь внутри этой ветви нужно разветвиться по $x_2 \leq 26$ и $x_2 \geq 27$, внутри этих ветвей получим значения целевой функции меньшие 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 29. 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:

$$u_+ = \{v \in V \mid (u, v) \in E\}$$

$$u_- = \{v \in V \mid (v, u) \in E\}$$

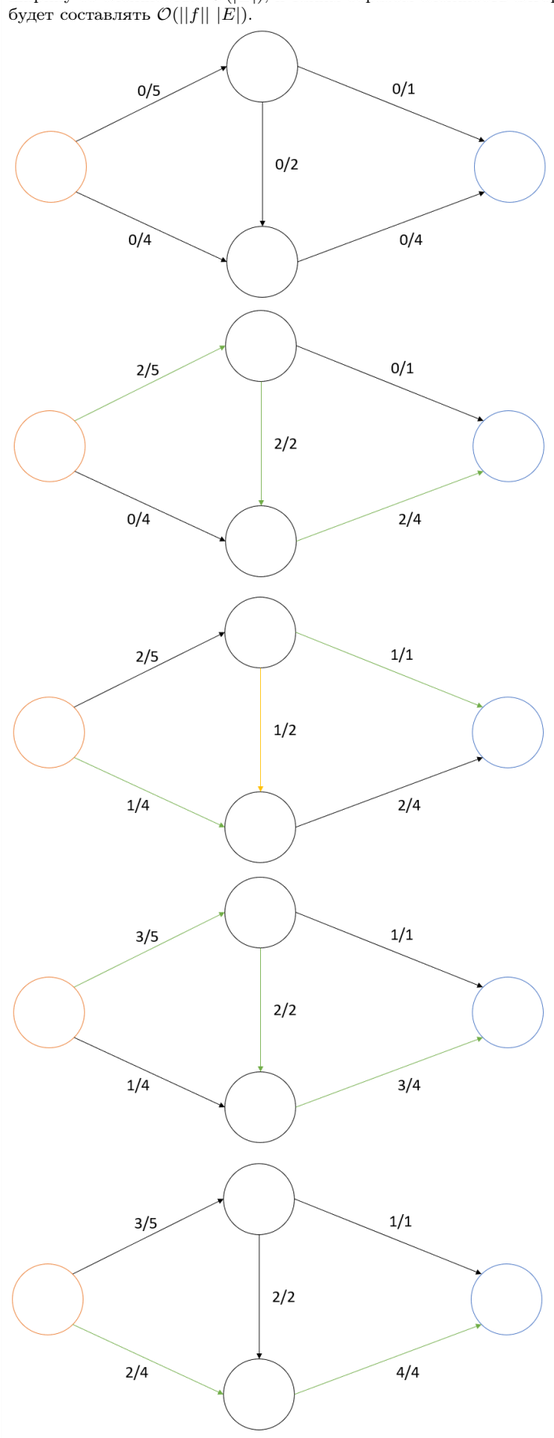
• f — **поток** в G, т.е. вектор, $|f| = |E|$, для которого выполнены условия:

1. $f(e) \in [0, c(e)], e \in E$
2. $\forall u \in V \setminus \{s, t\}$, $f_+(u) = \sum\limits_{v\in u_+} f(u, v) = \sum\limits_{v\in u_-} f(v, u) = f_-(u)$.

• $||f|| = f_+(s)$ — **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 сходится при условии целочисленности пропускной способности c 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 $\mathcal{O}(|E|)$, и таким образом сложность алгоритма будет составлять $\mathcal{O}(||f||\ |E|)$.



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

osn 33. 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 стержне.
$u(x, t)$ — 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.**

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

• **Вторая краевая задача.**

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

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

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

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

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

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

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

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

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

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:

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

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

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

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

dop 28. Комбинаторные методы нахождения оптимального пути в графе.

Определения

• Взвешенный орграф $G = (V, E, c)$ называется *сетью* (c — функция, определяющая вес ребра).

• Ориентированный маршрут называется *путем*.

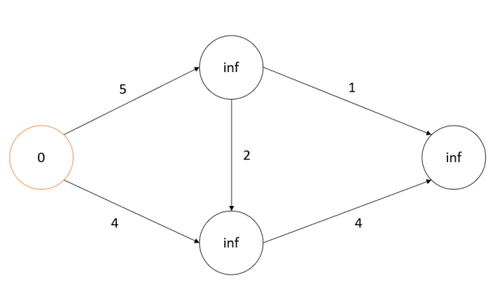
• Пусть P — некоторый (v, w)-путь: $v = v0 \to v_1 \to \cdots \to v_k = w$. Тогда $l(P) = c(e_1) + c(e_2) + \cdots + c(e_k)$ называется длиной пути P, где e_i — ребро перехода от вершины v_{i-1} к вершине v_i.

• (u, w)-путь с наименьшей длиной называется *кратчайшим*.

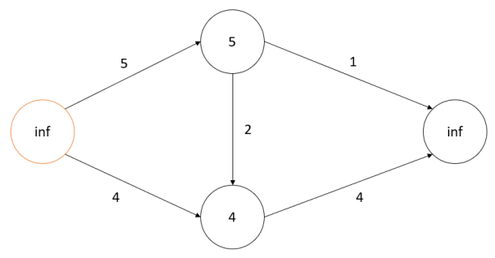
• Задача о кратчайшем пути между фиксированными вершинами: в заданной сети G с двумя выделенными вершинами s и t найти кратчайший (s, t)-путь.

Алгоритм Беллмана-Форда. Сложность $\mathcal{O}(|V| + |E|)$.
Пусть мы хотим найти длину кратчайшего пути из s в остальные вершины. Обозначим через a_{kp} длину кратчайшего пути из s в p, состоящего из k дуг, или inf, если такого пути не существует.
Инициализация. $a_{0s} = 0$, $a_{0p} = \text{inf}$ для всех вершин $p! = k$.
Шаг алгоритма. Зная все минимальные длины путей из $k - 1$ дуг, посчитать минимальную длину пути из k дуг можно, перебрав все вершины, из которых имеются дуги, идущие в данную. Для всех вершин p: $a_{kp} = min\{a_{k-1,p'} + c((p', p)) \mid p' \in V, (p', p) \in E\}$.
Шаг повторяется $|V| - 1$ раз, так как если путь содержит больше дуг, то в нем точно имеется цикл, который можно выбросить. На практике нет необходимости хранить всю матрицу целиком, нужно хранить лишь три строки - ранее вычисленную a_{k-1}, вычисляемую сейчас a_k, и строку с ответами, которая обновляется на кажом шаге: $ans_p = min\{ans_p, a_{kp}\}$.

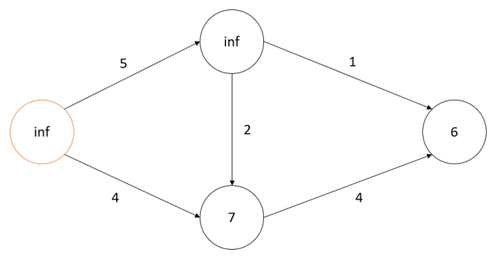
K = 0



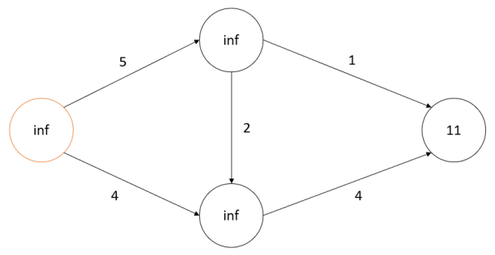
K = 1



K = 2

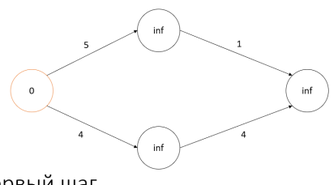


K = 3

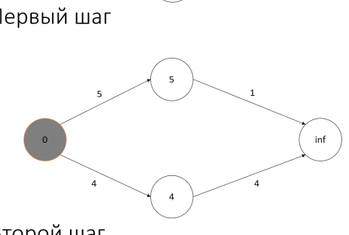


Алгоритм Дейкстры. Сложность варьируется (см. ниже).
Каждой вершине из V сопоставим метку — минимальное известное расстояние от этой вершины до s. Алгоритм работает пошагово — на каждом шаге он посещает одну вершину и пытается уменьшать метки. Работа алгоритма завершается, когда все вершины посещены.
Инициализация. Метка самой вершины s полагается равной 0, метки остальных вершин — inf. Это отражает то, что расстояния от s до других вершин пока неизвестны. Все вершины графа помечаются как непосещённые.
Шаг алгоритма. Если все вершины посещены, алгоритм завершается. В противном случае, из ещё не посещённых вершин выбирается вершина u, имеющая минимальную метку. Обновим метки соседних с u вершин - $\forall s : (u, s) \in E : label_s = min\{label_s, label_u + c((u, s))\}$ Пометим вершину u посещенной.
Сложность алгоритма варьируется в зависимости от того как хранить множество непосещенных вершин для удобства получения вершины с минимальной веткой. При хранении множества как списка, упорядоченного по убыванию меток, сложность алгоритма состявляет $\mathcal{O}(|V|^2)$. При использовании двоичной кучи сложность уменьшается до $\mathcal{O}((|E| + |V|) \log |V|)$.

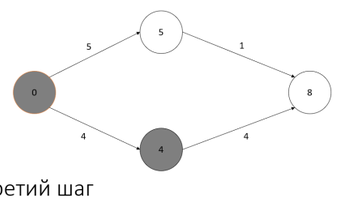
Начальное состояние



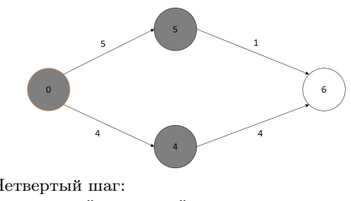
Первый шаг



Второй шаг



Третий шаг



Четвертый шаг:
+ серенький последний круг

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

dop 26. 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, у 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** $(\ldots, p_j, p_{j+1}, p_{j+2}, \ldots)$ расположены между её инструкциями $(\ldots, I_j, I_{j+1}, \ldots)$ 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 точке программы p_j перед инструкцией I_j (**входным состоянием**, $In[I_j]$) и состоянием в точке 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 до выходного состояния) f_{I_j} и передаточная функция обратного обхода (от выходного до входного состояния) $f^b_{I_j}$. 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 $In[I_j] = f^b_{I_j}(Out[I_j])$ при обратном.

• Для ББ B из инструкций $I_1, \ldots, I_n$ 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: $f_B(x) = f_{I_n}(f_{I_{n-1}}(\ldots f_{I_1}(x)\ldots)) = (f_{I_1} \circ f_{I_2} \circ \cdots \circ f_{I_n})(x)$, или $f_B = f_{I_1} \circ f_{I_2} \circ \cdots \circ f_{I_n}$ и, соответственно, $f^b_B = f^b_{I_n} \circ f^b_{I_{n-1}} \circ \cdots \circ f^b_{I_1}$.

Итого, при прямом обходе: $Out[B] = f_B(In[B])$; 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 операндов которой является переменная x.
- Определение d **достигает** точки p, если существует путь от точки, непосредственно следующей 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, $f_I(x) = gen_I \cup (x - killI)$, где x — состояние во входной точке инструкции I.
Пусть базовый блок B содержит n инструкций, каждая из которых имеет передаточную функцию $f_i(x) = gen_i \cup (x - kill_i)$, $i = 1, 2, \ldots, n$. Тогда передаточная функция для базового блока B может быть записана как $f_B(x) = gen_B \cup (x - kill_B)$, где $kill_B = kill_1 \cup kill_2 \cup \cdots \cup kill_n$ и $gen_B = gen_n \cup (gen_{n-1} - kill_n) \cup \cdots \cup (gen_1 - kill_2 - kill_3 - \cdots - kill_n)$. Таким образом, для каждого базового блока B_i можно выписать уравнение: $Out[B_i] = f_B(In[B_i])$.
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:

$$In[B_i] = \bigcup_{P \in Pred(B_i)} (gen_P \cup (In[P] - kill_P),\ i = 1, 2, \ldots n$$

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 $\leqslant$: $x \leqslant y$ тогда и только тогда, когда $x \wedge y = x$.

• **Структурой потока данных** называется четверка $<D, F, L, \wedge>$, 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**: $<Forward, GK, Def, \cup>$, где GK — семейство передаточных функций вида gen-kill, Def — множество определений переменных.

• Структура потока данных $<D, F, L, \wedge>$ называется **монотонной**, если $\forall x, y \in L,\ \forall f \in F\ (x \leqslant y) \Rightarrow f(x) \leqslant f(y)$.

• Структура потока данных $<D, F, L, \wedge>$ называется **монотонной** (определение эквивалентно предыдущему), если $\forall x, y \in L,\ \forall f \in F\ f(x \wedge y) \leqslant f(x) \wedge f(y)$.

• Структура потока данных $<D, F, L, \wedge>$ 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
(Вообще, никто не называет это методом неподвижной точки, но видимо имелось ввиду это.)
Общий итеративный алгоритм решения задачи анализа потока данных:
Вход: граф потока управления, структура потока данных $<D, F, L, \wedge>$, 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[ВХОД] = $v_{\text{ВХОД}}$;
2) **for** (каждый базовый блок B, отличный от входного) OUT$[B] = \top$;
3) **while** (внесены изменения в OUT)
4) **for** (каждый базовый блок B, отличный от входного) {
5) IN$[B] = \wedge_{P - \text{предшественник } B}OUT[P]$;
6) OUT$[B] = f_B$(IN$[B]$);
}

а) Итеративный алгоритм для прямой задачи потока данных

1) IN[ВЫХОД] = $v_{\text{ВЫХОД}}$;
2) **for** (каждый базовый блок B, отличный от выходного) IN$[B] = \top$;
3) **while** (внесены изменения в IN)
4) **for** (каждый базовый блок B, отличный от выходного) {
5) OUT$[B] = \wedge_{S - \text{преемник } B}IN[S]$;
6) IN$[B] = f_B$(OUT$[B]$);
}

б) Итеративный алгоритм для обратной задачи потока данных

Проходим по всем блокам и вычисляем множества In и Out для каждого блока пока что-то меняется. **(Когда ничего не меняется это, видимо неподвижная точка, в драгонбуке называется фиксированной)**
Для достигающих определений см. таблицу. Там еще есть другие анализы на всякий случай.

	Достигающие определения
Область определения	Множества определений
Направление	Прямое
Передаточная функция	$gen_B \cup (x - kill_B)$
Граничное условие	OUT[ВХОД] $= \emptyset$
Оператор сбора $\wedge$	$\cup$
Уравнения	$OUT[B] = f_B(IN[B])$ $IN[B] = \wedge_{P \in pred(B)} OUT[P]$
Инициализация	OUT[B] $= \emptyset$

[Ульман и др., *Компиляторы: принципы, технологии и инструментарий (Драгонбук)*, с. 9.2 - 9.3]

Список литературы

Evergreen. *Какая-то ссылка про подходы к разработке.* URL: https://evergreens.com.ua/ru/articles/software-development-metodologies.html.

А. М. Денисов, А. В. Разгулин. *Обыкновенные дифференциальные уравнения, часть 1.* 2009. URL: https://drive.google.com/file/d/1piqpfimj19np-256y8QIstlzqBJdG-Cs/view?usp=sharing.

А. Н. Тихонов, А. А. Самарский. *Уравнения математической физики.* 1977. URL: https://docviewer.yandex.ru/view/102697704/?*=b%2Ft9LX22Pg1KxdGBFlY305K%2B7pN7InVybCI6InlhLWRpc2stcHVibGljOi8vbGpHSi9LVVhTUXRUUmlwUk9aWWdtajhqUXpzTzRBWWVWOHBqM1NDamU2RE9PVnpxZnJkM3kwem1kY01WYytQU3EvSjZi2F0L7RgtC%2B0LovWzUg0YHQtdC8L10g0KPRgNCw0LLQvdC10L3QuNGPINC80LDRgtC10LzQsNGC0LjRh9C10YHQutC%2B0Lkg0YTQuNC30LjQutC4LzEuINCa0L3QuNCz0Lgv0JTQvtC%2F0L7Qu9C90LjRgtC10LvRjNC90L4v0KLQuNGF0L7QvdC%2B0LIg0JAu0J0uINCh0LDQvNCw0YDRgdC60LjQuSDQkC7QkC4g0KPRgNCw0LLQvdC10L3QuNGPINC80LDRgt2B0Lkg0YTQuNC30LjQutC4ICgxOTc3KS5kanZ1IiwidGl0bGUiOiLQotC40YXQvtC90L7QsiDQkC7QnS4g0KHQsNC80LDRgNGB0LrQuNC5INCQLtCQLiDQo9GA0LDQstC90LXQvdC40Y8g0LzQsNGC0LXQvNCw0YLQuNGH0LXRgdC60L7QuSDRhNC40LfQuNC60LggKDE5NzcpLmRqdnUiLCJub2lmcmFtZSI6ZmFsc2UsInVpZCI6IjEwMjY5NzcwNCIsInRzIjoxNj3D%3D.

Бахтин, В. А. *Презы по распределенным системам.* URL: http://dvmh.keldysh.ru/projects/ros2021/files?sort=created_on%3Adesc%2Cfilename.

В. А. Ильин В. А. Садовничий, Б. Х. Сендов. *Математический анализ, продолжение курса.* 1987. URL: https://docs.yandex.ru/docs/view?url=ya-disk-public%3A%2F%2FljGJ%2FKUXSQtTRipROZYgmj8jQzsO4AYeV8pj3SCje6DOOVzqfrd3y0zmdcMVc%2BPSq%2FJ6bpmRyOJonT3VoXnDag%3D%3D%3A%2F2%20%D0%BA%D1%83%D1%80%D1%81%2F%D0%9C%D0%B0%D1%82%D0%B0%D0%BD%20%D0%B8%20%D0%A2%D0%A4%D0%9A%D0%9F%2F1.%20%D0%9A%D0%BD%D0%B8%D0%B3%D0%B8%2F1.%20%D0%94%D0%B5%D0%B9%D1%81%D1%82%D0%B2%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9%20%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7%2F%5B%D1%81%D0%B8%D0%BD%D0%B8%D0%B9%5D%20%D0%98%D0%BB%D1%8C%D0%B8%D0%BD%2C%20%D0%A1%D0%B0%D0%B4%D0%BE%D0%B2%D0%BD%D0%B8%D1%87%D0%B8%D0%B9%2C%20%D0%A1%D0%B5%D0%BD%D0%B4%D0%BE%D0%B2%20(%D1%87.2)%2F%D0%98%D0%BB%D1%8C%D0%B8%D0%BD%2C%20%D0%A1%D0%B0%D0%B4%D0%BE%D0%B2%D0%BD%D0%B8%D1%87%D0%B8%D0%B9%2C%20%D0%A1%D0%B5%D0%BD%D0%B4%D0%BE%D0%B2%20-%20%D0%9C%D0%90.pdf&name=%D0%98%D0%BB%D1%8C%D0%B8%D0%BD%2C%20%D0%A1%D0%B0%D0%B4%D0%BE%D0%B2%D0%BD%D0%B8%D1%87%D0%B8%D0%B9%2C%20%D0%A1%D0%B5%D0%BD%D0%B4%D0%BE%D0%B2%20-%20%D0%9C%D0%90.pdf&nosw=1.

В. Н. Пильщиков В. Г. Абрамов, А. А. Вылиток. *Машина Тьюринга и алгоритмы Маркова. Решение задач. Учебно-методическое пособие.* 2016. URL: https://docs.yandex.ru/docs/view?url=ya-disk-public%3A%2F%2FljGJ%2FKUXSQtTRipROZYgmj8jQzsO4AYeV8pj3SCje6DOOVzqfrd3y0zmdcMVc%2BPSq%2FJ6bpmRyOJonT3VoXnDag%3D%3D%3A%2F1%20%D0%BA%D1%83%D1%80%D1%81%2F%5B1%20%D0%BF%D0%BE%D1%82%D0%BE%D0%BA%2C%201%20%D1%81%D0%B5%D0%BC.%5D%20%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%8B%20%D0%B8%20%D0%90%D0%AF%2F1.%20%D0%9A%D0%BD%D0%B8%D0%B3%D0%B8%2F%D0%9F%D0%B8%D0%BB%D1%8C%D1%89%D0%B8%D0%BA%D0%BE%D0%B2%20-%20%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%B0%20%D0%A2%D1%8C%D1%8E%D1%80%D0%B8%D0%BD%D0%B3%D0%B0%20%D0%B8%20%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%8B%20%D0%9C%D0%B0%D1%80%D0%BA%D0%BE%D0%B2%D0%B0(2016).pdf&name=%D0%9F%D0%B8%D0%BB%D1%8C%D1%89%D0%B8%D0%BA%D0%BE%D0%B2%20-%20%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%B0%20%D0%A2%D1%8C%D1%8E%D1%80%D0%B8%D0%BD%D0%B3%D0%B0%20%D0%B8%20%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D1%8B%20%D0%9C%D0%B0%D1%80%D0%BA%D0%BE%D0%B2%D0%B0(2016).pdf&nosw=1.

Г. Д. Ким, В. А. Ильин. *Линейная алгебра и аналитическая геометрия.* 2007. URL: https://docs.yandex.ru/docs/view?url=ya-disk-public%3A%2F%2FljGJ%2FKUXSQtTRipROZYgmj8jQzsO4AYeV8pj3SCje6DOOVzqfrd3y0zmdcMVc%2BPSq%2FJ6bpmRyOJonT3VoXnDag%3D%3D%3A%2F1%20%D0%BA%D1%83%D1%80%D1%81%2F%D0%90%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0%20%D0%B8%20%D0%B3%D0%B5%D0%BE%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%8F%20(%D0%9B%D0%B8%D0%BD%D0%B0%D0%BB)%2F1.%20%D0%9A%D0%BD%D0%B8%D0%B3%D0%B8%2F%5B%D0%A3%D1%87%D0%B5%D0%B1%D0%BD%D0%B8%D0%BA%5D%20%D0%9A%D0%B8%D0%BC%2C%20%D0%98%D0%BB%D1%8C%D0%B8%D0%BD%20(%D0%9B%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D0%B0%D1%8F%20%D0%B0%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0%20%D0%B8%20%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F%20%D0%B3%D0%B5%D0%BE%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%8F).pdf&name=%5B%D0%A3%D1%87%D0%B5%D0%B1%D0%BD%D0%B8%D0%BA%5D%20%D0%9A%D0%B8%D0%BC%2C%20%D0%98%D0%BB%D1%8C%D0%B8%D0%BD%20(%D0%9B%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D0%B0%D1%8F%20%D0%B0%D0%BB%D0%B3%D0%B5%D0%B1%D1%80%D0%B0%20%D0%B8%20%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F%20%D0%B3%D0%B5%D0%BE%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%8F).pdf.

Головин, И.Г. *Материалы по языкам программирования к госэкзамену.* URL: https://m.cs.msu.ru/index.php/s/N6FkcmFbxQkS8z9?path=%2F%D0%93%D0%BE%D0%BB%D0%BE%D0%B2%D0%B8%D0%BD%D0%98%D0%93%2F%D0%93%D0%9E%D0%A1-%D0%9A%D0%BE%D0%BD%D1%81%D1%83%D0%BB%D1%8C%D1%82%D0%B0%D1%86%D0%B8%D1%8F.

И. С. Рожков, А. С. Рыгин. *Почти наверное достаточное учебное пособие. Теория вероятностей и математическая статистика. Статбук.* 2020. URL: https://mandaloreultimate.github.io/stat-book/main.pdf.

И.В. Садовничая, Е.В. Хорошилова. *Определенный интеграл.* URL: https://docs.yandex.ru/docs/view?url=ya-disk-public%3A%2F%2FljGJ%2FKUXSQtTRipROZYgmj8jQzsO4AYeV8pj3SCje6DOOVzqfrd3y0zmdcMVc%2BPSq%2FJ6bpmRyOJonT3VoXnDag%3D%3D%3A%2F1%20%D0%BA%D1%83%D1%80%D1%81%2F%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9%20%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7%2F1.%20%D0%9A%D0%BD%D0%B8%D0%B3%D0%B8%2F%5B%D0%9C%D0%B5%D1%82%D0%BE%D0%B4%D0%B8%D1%87%D0%BA%D0%B8%5D%20%D0%A1%D0%B0%D0%B4%D0%BE%D0%B2%D0%BD%D0%B8%D1%87%D0%B0%D1%8F%20%D0%A5%D0%BE%D1%80%D0%BE%D1%88%D0%B8%D0%BB%D0%BE%D0%B2%D0%B0%20%D0%A4%D0%BE%D0%BC%D0%B5%D0%BD%D0%BA%D0%BE%2F5.%20%D0%A1%D0%B0%D0%B4%D0%BE%D0%B2%D0%BD%D0%B8%D1%87%D0%B0%D1%8F.%20%D0%9E%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9%20%D0%B8%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B0%D0%BB.pdf&name=5.%20%D0%A1%D0%B0%D0%B4%D0%BE%D0%B2%D0%BD%D0%B8%D1%87%D0%B0%D1%8F.%20%D0%9E%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9%20%D0%B8%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B0%D0%BB.pdf.

И.С. Ломов, А.А. Никитин. *Рукописные лекции по математическому анализу.* 2019.

Игнатьев, В. Н. *Лекция по формалкам 4 слайды.* URL: https://drive.google.com/file/d/1R1iQd8ZLr7NfFyhwMDN_bVhVmrXUnXnd/view?usp=sharing.

Ионкин, Н.И. *Лекции по курсу Численные методы.* 2015. URL: https://docs.yandex.ru/docs/view?url=ya-disk-public%3A%2F%2FljGJ%2FKUXSQtTRipROZYgmj8jQzsO4AYeV8pj3SCje6DOOVzqfrd3y0zmdcMVc%2BPSq%2FJ6bpmRyOJonT3VoXnDag%3D%3D%3A%2F4%20%D0%BA%D1%83%D1%80%D1%81%2F3%20%D0%BF%D0%BE%D1%82%D0%BE%D0%BA%2F%D0%A7%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%BD%D1%8B%D0%B5%20%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D1%8B%2F%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D0%B8%D1%87%D0%BA%D0%B0%2C%20%D0%BB%D0%B5%D0%BA%D1%86%D0%B8%D0%B8%2F%D0%A7%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%BD%D1%8B%D0%B5%20%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D1%8B.%20%D0%98%D0%BE%D0%BD%D0%BA%D0%B8%D0%BD%20(%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D0%B8%D1%87%D0%BA%D0%B0)%20(2015).pdf&name=%D0%A7%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%BD%D1%8B%D0%B5%20%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D1%8B.%20%D0%98%D0%BE%D0%BD%D0%BA%D0%B8%D0%BD%20(%D0%BC%D0%B5%D1%82%D0%BE%D0%B4%D0%B8%D1%87%D0%BA%D0%B0)%20(2015).pdf&nosw=1.

Кузнецов, С. Д. *Основы современных БД.* 2019. URL: https://drive.google.com/file/d/1uvAyzdQt9yHSusCXvrb7BZ_d1Yfh_D1Q/view?usp=sharing.

Курс распределенных систем. URL: https://parallel.ru/krukov/lec5.html.

Курс распределенных систем. URL: https://parallel.ru/krukov/lec7.html.

Лекции асм 1 потока. URL: http://asmcourse.cs.msu.ru/?page_id=2439.

Маш-Маш. *Машбук. Операционные системы. Конспект лекций.* 2017. URL: https://docs.yandex.ru/docs/view?url=ya-disk-public%3A%2F%2FljGJ%2FKUXSQtTRipROZYgmj8jQzsO4AYeV8pj3SCje6DOOVzqfrd3y0zmdcMVc%2BPSq%2FJ6bpmRyOJonT3VoXnDag%3D%3D%3A%2F2%20%D0%BA%D1%83%D1%80%D1%81%2F%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%2F1.%20%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5%20%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B%2F1.%20%D0%9A%D0%BD%D0%B8%D0%B3%D0%B8%20(%D0%B2%20%D1%81%D0%BE%D0%BE%D1%82%D0%B2%D0%B5%D1%82%D1%81%D0%B2%D0%B8%D0%B8%20%D1%81%D0%BE%20%D1%81%D0%BF%D0%B8%D1%81%D0%BA%D0%BE%D0%BC%20%D0%BB%D0%B8%D1%82-%D1%80%D1%8B%20%D1%81%20%D0%BB%D0%B5%D0%BA%D1%86%D0%B8%D0%B9)%2F2017.%20%D0%9C%D0%B0%D1%88%D0%B1%D1%83%D0%BA.pdf&name=2017.%20%D0%9C%D0%B0%D1%88%D0%B1%D1%83%D0%BA.pdf&nosw=1.

Презентации Гайсаряна. URL: https://docs.yandex.ru/docs/view?url=ya-disk-public%3A%2F%2FljGJ%2FKUXSQtTRipROZYgmj8jQzsO4AYeV8pj3SCje6DOOVzqfrd3y0zmdcMVc%2BPSq%2FJ6bpmRyOJonT3VoXnDag%3D%3D%3A%2F4%20%D0%BA%D1%83%D1%80%D1%81%2F3%20%D0%BF%D0%BE%D1%82%D0%BE%D0%BA%2F%D0%9A%D0%BE%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5%20%D0%BA%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D1%8F%D1%82%D0%BE%D1%80%D0%BE%D0%B2%2F%D0%9F%D1%80%D0%B5%D0%B7%D0%B5%D0%BD%D1%82%D0%B0%D1%86%D0%B8%D0%B8%2F2019.%20Slides%2Fmerged-konstr.pdf&name=merged-konstr.pdf&nosw=1.

Презентации к госам от маткиба. URL: https://mk.cs.msu.ru/index.php/%D0%94%D0%B8%D1%81%D0%BA%D1%80%D0%B5%D1%82%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0_(1-%D0%B9_%D0%BF%D0%BE%D1%82%D0%BE%D0%BA).

Презентация по освещению от ИИТа. URL: https://docplayer.com/32165509-Lokalnye-i-globalnye-modeli-osveshcheniya-model-fonga-zakraska.html.

Пупкин-Залупкин. *Напиши источник!* 1234. URL: https://zona.media/.

С. И. Гуров, Д. А. Кропотов-плохой-человек. *Запись лекций по курсу прикладная алгебра. Алгебраические основы кодирования и шифрования.* 2020. URL: https://drive.google.com/file/d/1fHnSPhxeavLbLsJ5pejMPVgLlPRhXf2n/view.

Смелянский, Р. Л. *Компьютерные сети, том 1.* URL: https://docs.yandex.ru/docs/view?url=ya-disk-public%3A%2F%2FljGJ%2FKUXSQtTRipROZYgmj8jQzsO4AYeV8pj3SCje6DOOVzqfrd3y0zmdcMVc%2BPSq%2FJ6bpmRyOJonT3VoXnDag%3D%3D%3A%2F3%20%D0%BA%D1%83%D1%80%D1%81%2F3%20%D0%BF%D0%BE%D1%82%D0%BE%D0%BA%2F%5B5%20%D1%81%D0%B5%D0%BC.%5D%20%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%B2%20%D1%81%D0%B5%D1%82%D0%B8%20%D0%AD%D0%92%D0%9C%2F%D0%9A%D0%BD%D0%B8%D0%B3%D0%B8%2F%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%8B%D0%B5%20%D1%81%D0%B5%D1%82%D0%B8.%20%D0%A2%D0%BE%D0%BC%201.%20%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B%20%D0%BF%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%87%D0%B8%20%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85.%20%D0%A0.%D0%9B.%20%D0%A1%D0%BC%D0%B5%D0%BB%D1%8F%D0%BD%D1%81%D0%BA%D0%B8%D0%B9.pdf&name=%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%8B%D0%B5%20%D1%81%D0%B5%D1%82%D0%B8.%20%D0%A2%D0%BE%D0%BC%201.%20%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B%20%D0%BF%D0%B5%D1%80%D0%B5%D0%B4%D0%B0%D1%87%D0%B8%20%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85.%20%D0%A0.%D0%9B.%20%D0%A1%D0%BC%D0%B5%D0%BB%D1%8F%D0%BD%D1%81%D0%BA%D0%B8%D0%B9.pdf.

— *Компьютерные сети, том 2.* URL: https://docs.yandex.ru/docs/view?url=ya-disk-public%3A%2F%2FljGJ%2FKUXSQtTRipROZYgmj8jQzsO4AYeV8pj3SCje6DOOVzqfrd3y0zmdcMVc%2BPSq%2FJ6bpmRyOJonT3VoXnDag%3D%3D%3A%2F3%20%D0%BA%D1%83%D1%80%D1%81%2F3%20%D0%BF%D0%BE%D1%82%D0%BE%D0%BA%2F%5B5%20%D1%81%D0%B5%D0%BC.%5D%20%D0%92%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%B2%20%D1%81%D0%B5%D1%82%D0%B8%20%D0%AD%D0%92%D0%9C%2F%D0%9A%D0%BD%D0%B8%D0%B3%D0%B8%2F%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%8B%D0%B5%20%D1%81%D0%B5%D1%82%D0%B8.%20%D0%A2%D0%BE%D0%BC%202.%20%D0%A1%D0%B5%D1%82%D0%B8%20%D0%B2%20%D0%AD%D0%92%D0%9C.%20%D0%A0.%D0%9B.%20%D0%A1%D0%BC%D0%B5%D0%BB%D1%8F%D0%BD%D1%81%D0%BA%D0%B8%D0%B9.pdf&name=%D0%9A%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D1%8B%D0%B5%20%D1%81%D0%B5%D1%82%D0%B8.%20%D0%A2%D0%BE%D0%BC%202.%20%D0%A1%D0%B5%D1%82%D0%B8%20%D0%B2%20%D0%AD%D0%92%D0%9C.%20%D0%A0.%D0%9B.%20%D0%A1%D0%BC%D0%B5%D0%BB%D1%8F%D0%BD%D1%81%D0%BA%D0%B8%D0%B9.pdf.

IT-сообщество. *Какие-то ссылки про БД.* https://habr.com/ru/post/254773/ ; https://info-comp.ru/database-normalization.

— *Статьи про сортировку на Хабре.* https://habr.com/ru/post/204600/ ; https://codelessons.ru/cplusplus/algoritmy/puzyrkovaya-sortirovka-v-c-glavnye-momenty.html ; https://habr.com/ru/post/188010/ ; https://habr.com/ru/post/181271/ ; http://mindhalls.ru/quick-sort-c-cpp/ ;

Ульман, Д. и др. *Компиляторы: принципы, технологии и инструментарий (Драгонбук).* ЛитРес, 2022. ISBN: 9785041399252. URL: https://books.google.ru/books?id=PH11DwAAQBAJ.

Шилдт, Герберт. *C++. Шаг за шагом.* 2013. URL: https://drive.google.com/file/d/1-fCB_A7zkX9YYSGyEnkDpOogTYgUcQV0/view?usp=sharing.