Тема 1. Введение в дисциплину. Основные понятия

Определение и свойства алгоритма

Алгоритм – это конечная, понятная и точная последовательность действий (операций), выполнение которых приводит к решению поставленной задачи.

Основные свойства (качества) алгоритма:

1. **Дискретность** – алгоритм состоит из отдельных (дискретных) шагов, каждый из которых выполняется за конечное время.
2. **Понятность (детерминированность)** – каждый шаг алгоритма однозначно определён и понятен исполнителю (человеку, компьютеру и т. д.).
3. **Результативность** – при выполнении алгоритма за конечное число шагов достигается требуемый результат или выводится сообщение о невозможности решения.
4. **Массовость** – алгоритм применим к целому классу однотипных задач.
5. **Конечность** – алгоритм должен завершаться за конечное время.

Концепция типов данных: синтаксис, семантика, прагматика

При описании языков программирования и типов данных обычно рассматривают следующие аспекты:

1. **Синтаксис** – форма представления данных и управляющих конструкций (например, правила записи литералов, имён переменных, операторов языка).
2. **Семантика** – смысловое содержание синтаксически корректных конструкций (как именно данные обрабатываются, как трактуются операторы).
3. **Прагматика** – практическая сторона использования языка или типа данных (эффективность, удобство в решении конкретных задач, применимость).

Уровни представления данных в программных приложениях

1. **Логический уровень** – описывает, какие именно данные хранятся и как они logically организованы (структуры данных «как есть»).
2. **Физический уровень** – описывает, как данные фактически размещаются в памяти (или на дисках, в файлах и т. д.).
3. **Абстрактный уровень (иногда выделяют отдельно)** – пользователю видны только операции над структурой данных, а реализация скрыта (абстракция данных).

Понятие структуры данных и ее назначение

Структура данных – это способ организации и хранения данных, обеспечивающий удобное выполнение операций над ними (добавление, удаление, поиск, сортировка и т.

д.). Назначение структур данных – обеспечить эффективную обработку данных с точки зрения времени и/или памяти.

Какие структуры данных называют линейными

Линейные структуры данных – это структуры, в которых каждый элемент (кроме первого и последнего) имеет не более двух «соседей», а сами элементы образуют единую последовательность. Примеры: массив, список, стек, очередь и т. п.

Классификация структур данных

1. По способу доступа к элементам:
    - **Линейные**: массив, список, очередь, стек и пр.
    - **Иерархические (древовидные)**: деревья.
    - **Сетевые (графовые)**: графы.
2. По реализации (статическая или динамическая).
3. По виду хранимых данных (простые/сложные, гомогенные/гетерогенные).

Понятие, свойства и способы описания алгоритмов

Способы описания (представления) алгоритмов:

- Словесно-формульный (текстовое описание).
- Блок-схема (графическое представление).
- Псевдокод (набор инструкций, напоминающих язык программирования).
- Непосредственно программный код.

Корректность алгоритма – алгоритм корректен, если для любых входных данных из области определения он даёт правильный результат (или корректное сообщение о невыполнимости задачи).

Спецификация алгоритма

Спецификация алгоритма определяет:

- Какие данные (формат/тип входных данных) поступают на вход.
- Каковы ожидаемые выходные данные и в каком формате они выдаются.
- Допустимые ограничения на входные данные (размер, тип и т.д.).
- Пред- и постусловия (что гарантируется до и после выполнения алгоритма).

Понятие модели данных, ее назначение

Модель данных – формальный способ описания данных, определяющий, как данные и взаимосвязи между ними должны быть структурированы и каким образом с ними можно работать. Назначение – упростить понимание, проектирование и реализацию структур данных.

Поэтапная разработка программ

1. Анализ задачи, формирование требований.
2. Разработка алгоритма, выбор структуры данных.
3. Кодирование (программная реализация).
4. Тестирование и отладка.
5. Внедрение и сопровождение.

Тема 2. Анализ вычислительной сложности алгоритмов

Ресурсы программы и назначение анализа эффективности

При анализе эффективности алгоритмов рассматривают затраты на:

1. **Время (количество выполняемых операций / шагов)**.
2. **Память (объём используемой памяти)**.
3. (Иногда) другие ресурсы: операции ввода-вывода, обращения к сети и т. д.

Назначение анализа эффективности – оценить, насколько «быстро» или «выгодно по памяти» будет работать алгоритм при увеличении размера входных данных.

Критерии оценки эффективности программы

1. **Вычислительная (временная) сложность** – как меняется количество операций (или время выполнения) при росте объёма входных данных.
2. **Пространственная сложность** – как изменяется объём оперативной памяти, необходимой для выполнения, при росте размера входных данных.

Что определяет размер задачи

Размер задачи часто трактуют как:

- Число элементов входных данных (n).
- Число бит, требуемых для представления данных, и т. д.

Чаще всего под **n** подразумевают число обрабатываемых элементов.

Вычислительная сложность алгоритма

Вычислительная сложность – функция, которая показывает зависимость количества выполняемых алгоритмом операций от размера входных данных (n).

Асимптотический анализ порядка роста вычислительной сложности алгоритмов

Асимптотический анализ – способ описания поведения функции (сложности) при очень больших n. Используются специальные **асимптотические обозначения (нотации)**.

Асимптотические обозначения (Big-O, Θ, Ω, o, ω)

1. **O(...)** (Big-O) – верхняя граница роста.
2. **Θ(...)** (Тета) – точная (двухсторонняя) оценка: и верхняя, и нижняя граница.
3. **Ω(...)** (Омега) – нижняя граница.
4. **o(...)** (малое o) – «строгая» верхняя граница (более узкая, чем O).

5. **ω(...)** (малое омега) – «строгая» нижняя граница.

Свойства асимптотических функций

- Если f(n)∈O(g(n))f(n) \in O(g(n))f(n)∈O(g(n)) и g(n)∈O(h(n))g(n) \in O(h(n))g(n)∈O(h(n)), то f(n)∈O(h(n))f(n) \in O(h(n))f(n)∈O(h(n)).
- Сумма функций: f(n)+g(n)∈O(max(f(n),g(n)))f(n)+g(n) \in O(\max(f(n), g(n)))f(n)+g(n)∈O(max(f(n),g(n))).
- Произведение функций: f(n)⋅g(n)∈O(f(n)⋅g(n))f(n)\cdot g(n) \in O(f(n)\cdot g(n))f(n)⋅g(n)∈O(f(n)⋅g(n)).
- И др.

Классы функций сложности алгоритмов (распространённые)

- **Константная**: O(1)O(1)O(1).
- **Логарифмическая**: O(logn)O(\log n)O(logn).
- **Линейная**: O(n)O(n)O(n).
- **Линейно-логарифмическая**: O(nlogn)O(n \log n)O(nlogn).
- **Квадратичная**: O(n2)O(n^2)O(n2).
- **Кубическая**: O(n3)O(n^3)O(n3).
- **Полиномиальная**: O(nk)O(n^k)O(nk).
- **Экспоненциальная**: O(2n)O(2^n)O(2n).
- **Факториальная**: O(n!)O(n!)O(n!).

Способы оценки вычислительной сложности алгоритмов

1. **Теоретический анализ** (по шагам с помощью операций «счётных» – подсчёт количества базовых операций).
2. **Экспериментальный анализ** (замеры времени работы программы с разным n).

Оценки эффективности в наилучшем, наихудшем, среднем случаях

- **Наихудший (Worst-case)** – учитывается сценарий, когда алгоритму требуется максимальное количество шагов.
- **Наилучший (Best-case)** – минимальное количество шагов.
- **Средний (Average-case)** – статистический средний (например, равновероятное распределение входных данных).

Применяется при выборе оптимального алгоритма для конкретных условий (если данные чаще «случайны» – интересует средняя оценка, если нужно гарантировать результат – смотрим на худший случай).

Тема 3. Рекурсивные алгоритмы

Понятие рекурсивного алгоритма

Рекурсивный алгоритм – алгоритм, в котором функция или подпрограмма вызывает саму себя (непосредственно или опосредованно) с упрощённым вариантом исходной задачи, пока не достигнет базового (граничного) случая.

Виды рекурсии

1. **Линейная** – когда за один рекурсивный вызов происходит только один дальнейший вызов.
2. **Каскадная (древовидная)** – когда каждый вызов порождает несколько рекурсивных вызовов (количество растёт как дерево).
3. **Удалённая** – когда один вызов функции идёт через другие функции или уровни (A вызывает B, B вызывает C, C вызывает A).

Шаг рекурсии, глубина рекурсивных вызовов

- **Шаг рекурсии** – переход от одной рекурсивной «итерации» к следующей (часто связан с упрощением задачи).
- **Глубина рекурсии** – максимальное число вложенных рекурсивных вызовов до достижения базового случая.

Условие завершения рекурсии

Базовый (граничный) случай, при котором дальнейших рекурсивных вызовов не производится.

Стек рекурсивных вызовов

При рекурсии среда выполнения (например, программа на C/C++/Java) использует стек вызовов, куда сохраняются текущие локальные переменные, адрес возврата и т. п.

Анализ рекурсивных алгоритмов, рекуррентное соотношение

Для оценки рекурсивных алгоритмов используют **рекуррентное соотношение**.
Пример: время работы $T(n)=T(n-1)+O(1)$ $T(n) = T(n-1) + O(1)$ $T(n)=T(n−1)+O(1)$ (линейная рекурсия).

Методы решения рекуррентных соотношений

1. **Метод подстановки** – предполагаем форму решения и проверяем, корректируя коэффициенты.
2. **Метод рекурсивного дерева** – рисуем дерево вызовов и суммируем количество операций на уровнях дерева.

3. **Основной метод** (Master theorem) – применяется к рекуррентным соотношениям вида $T(n)=aT(n/b)+f(n)T(n)=aT(n/b)+f(n)T(n)=aT(n/b)+f(n)$.

Тема 4. Простые алгоритмы сортировки

Определение, виды и характеристики алгоритмов сортировки

Сортировка – упорядочивание элементов массива (или другой структуры) по возрастанию (убыванию) некоторого ключа.

Ключевые характеристики:

- Количество сравнений.
- Количество обменов (перестановок элементов).
- Временная и пространственная сложность.

Основные операции с сортируемыми данными

1. **Сравнение** двух элементов.
2. **Обмен (swap)** элементов местами.
3. **Присваивание/копирование**.

Инвариант цикла, назначение, свойства и его применение

Инвариант цикла – условие (свойство), которое истинно при каждом проходе цикла. Используется для доказательства корректности и понимания, какие элементы уже «отсортированы»/«на месте».

Простые алгоритмы сортировки

1. **Сортировка простым обменом (Bubble Sort)**
    - На каждом проходе «всплывают» большие элементы в конец массива.
    - Сложность в худшем случае O(n2)O(n^2)O(n2), в лучшем – O(n)O(n)O(n) (если массив уже отсортирован и проверяется флаг перестановок).
2. **Сортировка простым выбором (Selection Sort)**
    - За каждый проход выбирается минимальный элемент из неотсортированной части и меняется местами с первым в этой части.
    - Сложность O(n2)O(n^2)O(n2) в худшем и лучшем случае.
3. **Сортировка простыми вставками (Insertion Sort)**
    - На каждом шаге берётся очередной элемент и «вставляется» в уже отсортированную часть массива.
    - В худшем случае O(n2)O(n^2)O(n2), в лучшем (почти отсортированные данные) – O(n)O(n)O(n).
4. **Условие Айверсона** для Bubble Sort
    - Позволяет прервать проход, если за очередную итерацию не было никаких обменов, значит массив уже отсортирован.

Сортировки за линейное время

- **Counting Sort (Сортировка подсчётом)** – эффективна, когда диапазон ключей относительно невелик; сложность O(n+k)O(n + k)O(n+k), где k – размер диапазона ключей.
- **Bucket Sort**, **Radix Sort** и др. – их эффективность также во многом зависит от структуры ключей.

Тема 5. Нетривиальные алгоритмы сортировки

Шейкерная сортировка (Shaker-sort)

- Модификация Bubble Sort, в которой «пробега» массива идут поочерёдно слева направо и справа налево.
- При определённых массивах может работать быстрее обычного Bubble Sort.
- Сложность в худшем случае остаётся O(n2)O(n^2)O(n2); в лучшем – O(n)O(n)O(n) (если проверять флаг перестановок).

Быстрая сортировка (Quick Sort)

- Разработана Т. Хоаром.
- Идея: «разделяй и властвуй» – выбирается опорный элемент (pivot), массив делится на две части: элементы меньше pivot и элементы больше pivot. Рекурсивно сортируем подмассивы.
- Сложность:
    - В **лучшем** случае (равномерный раскол) – O(nlogn)O(n \log n)O(nlogn).
    - В **худшем** случае (неудачно выбран pivot, массив почти не делится) – O(n2)O(n^2)O(n2).

Сортировка методом Хоара

Обычно именно её и называют Quick Sort. Принцип такой же: выбор pivot, разделение (partition), затем рекурсивная сортировка подмассивов.

Сортировка Шелла (Shell sort)

- Основана на идее «сортировок по убывающему шагу (gap)».
- Сначала сравниваются элементы на расстоянии gap, а затем gap уменьшается (например, последовательностью ⌊n/2⌋\lfloor n/2 \rfloor⌊n/2⌋, ⌊n/4⌋\lfloor n/4 \rfloor⌊n/4⌋ и т. п.).
- Итоговая сложность зависит от выбора последовательности шагов, но классически считается от O(n1.3)O(n^{1.3})O(n1.3) до O(n2)O(n^2)O(n2) в худшем случае. Средняя обычно лучше, чем у простых сортировок.

Пирамидальная сортировка (Heap-sort)

- Использует структуру **пирамиды (кучи)** – двоичная куча (binary heap).
- Построение кучи – O(n)O(n)O(n).
- Далее n раз извлекается максимум (или минимум) и восстанавливается куча – каждое извлечение O(logn)O(\log n)O(logn).
- Общая сложность – O(nlogn)O(n \log n)O(nlogn) в худшем, среднем, лучшем случаях.

Турнирная сортировка

- По сути – это сортировка с помощью «турнирного дерева».
- Сложность – примерно O(nlogn)O(n \log n)O(nlogn). Применяется для потоковых данных и т. п.

Сортировка подсчетом (Counting-sort)

- Используется для целых чисел ограниченного диапазона [0..k].
- Комплексная сложность O(n+k)O(n + k)O(n+k).
- Эффективна, когда k «не слишком велико» по сравнению с n.

Сортировка слиянием (Merge-sort)

- Принцип «разделяй и властвуй» – разбиваем массив пополам, рекурсивно сортируем подмассивы, затем сливаем их в один.
- Время O(nlogn)O(n \log n)O(nlogn) в худшем, лучшем и среднем случаях.
- Требуется дополнительная память O(n)O(n)O(n).

Варианты:

- Прямая (простая) сортировка слиянием.
- Естественная сортировка слиянием (когда уже есть «отсортированные подпоследовательности» в исходных данных).
- Двухфазное и однофазное слияние, двухпутевое и многопутевое слияния.

Тема 6. Алгоритмы поиска в массивах

Определение, виды и характеристики алгоритмов поиска

Поиск – операция, в ходе которой определяется, присутствует ли некоторый элемент (ключ) в массиве/списке, и если да, то на какой позиции.

Линейный поиск

- Суть: просматриваем элементы массива последовательно, пока не найдём нужный.
- В худшем случае O(n)O(n)O(n), в среднем – O(n)O(n)O(n), в лучшем – O(1)O(1)O(1) (если первый элемент – искомый).

Бинарный поиск

- Применим к **отсортированным** массивам.
- Суть: на каждом шаге сравниваем ключ с серединным элементом, отбрасываем половину массива и продолжаем.
- Сложность – O(logn)O(\log n)O(logn) в худшем, лучшем, среднем случаях.

Интерполяционный поиск

- Аналог бинарного, но берём не середину, а точку «предполагаемого положения» по формуле интерполяции.
- Средняя сложность лучше, чем у бинарного (O(loglogn)O(\log \log n)O(loglogn) в идеальных случаях), но в худшем – может быть O(n)O(n)O(n).

Фибоначчиев поиск

- Ещё один вариант поиска в отсортированных массивах, использует числа Фибоначчи для определения точек сравнения.
- Сложность аналогична бинарному поиску, порядка O(logn)O(\log n)O(logn).

Алгоритм поиска по бинарному дереву

- Если массив реализован в виде **бинарного дерева поиска (BST)**, то поиск идет по вершинам дерева.
- Средний случай O(logn)O(\log n)O(logn), если дерево сбалансировано; худший – O(n)O(n)O(n), если дерево выродилось в цепочку.

Тема 7. Алгоритмы поиска по образцу (строковый поиск)

Алгоритм прямого поиска по образцу (Naive)

- Сопоставляем шаблон (pattern) со строкой (text), сдвигаясь по одному символу.
- Худший случай O(nm)O(nm)O(nm), где n – длина текста, m – длина шаблона.

Метод Кнута-Морриса-Пратта (КМП)

- Идея: при несовпадении не нужно возвращаться назад в тексте, используем предварительно рассчитанный массив «длин префикса-суффикса».
- Вычислительная сложность – O(n+m)O(n+m)O(n+m).
- Эффективен при больших текстах и длинных шаблонах.

Алгоритм Бойера-Мура

- Сравнение начинается с конца шаблона. Использует «таблицу смещений» (bad character, good suffix).
- На практике один из самых быстрых для поиска подстроки в тексте.
- Худшая сложность – O(nm)O(nm)O(nm), средняя – близка к O(n)O(n)O(n).

Алгоритм Рабина-Карпа

- Используется «хеш» (контрольная сумма) для подстрок текста.
- Теоретически худший случай O(nm)O(nm)O(nm), на практике (при хорошем хеш-функции) среднее ~ O(n+m)O(n+m)O(n+m).

Тема 8. Линейные списки

Свойства структур данных

- **Линейный список** – элементы упорядочены последовательно, операции добавления/удаления обычно либо в начало, либо в конец, либо в произвольное место, если доступ к указателям.

Линейный список и его свойства

- Последовательность узлов, каждый узел хранит **значение** и (как минимум) ссылку на следующий элемент.
- Нет «прямого» индекса как в массиве, приходится итерироваться от начала.

Способы реализации линейных списков

1. **Статическая** – на базе массива. Достоинство: быстрый доступ по индексу; недостаток: фиксированный размер.
2. **Динамическая** – с помощью узлов и указателей (связные списки). Достоинство: лёгкое изменение размера; недостаток: нет случайного доступа.

Разновидности линейных списков: стек, очередь, дек

1. **Стек (Stack)** – принцип LIFO (последним пришёл, первым вышел). Операции `push` и `pop`.
2. **Очередь (Queue)** – принцип FIFO (первым пришёл, первым вышел). Операции `enqueue` (положить) и `dequeue` (взять).
3. **Дек (Deque)** – двухсторонняя очередь, элементы можно добавлять/удалять с обоих концов.

Обратная польская запись (ОПЗ)

- Способ записи арифметических выражений без скобок.
- Активно используется стек для преобразования выражений и вычисления ОПЗ.

Примеры использования стеков

- Парсер выражений (скобки), отмена операций (undo), системы вызовов подпрограмм.

Линейный односвязанный список

- Узел: содержит информационное поле и указатель на следующий узел.
- Операции вставки, удаления и поиска по значению обычно требуют прохода от головы списка до нужной позиции.

Двунаправленный связанный список

- Узел содержит указатели как на следующий, так и на предыдущий элемент.
- Удобнее удалять/вставлять в середине, но требуется чуть больше памяти и аккуратнее работать с указателями.

Кольцевые (циклические) списки

- Последний узел «ссылается» на первый. Позволяет «по кругу» обходить узлы.

Мультисписки

- Структура, в которой один элемент (один узел) может входить в несколько списков, имея несколько «следующих» ссылок.

Тема 9. Хеш-таблицы и алгоритмы хеширования и поиска с применением таблиц

Поиск с использованием хеш-таблицы, вычислительная сложность

- Идея: ключ пропускается через хеш-функцию, которая выдаёт индекс в массиве.
- В среднем **поиск** и **вставка** занимают O(1)O(1)O(1).
- В худшем случае (коллизии) – O(n)O(n)O(n).

Хеш-функция, её назначение, результат

- **Хеш-функция** преобразует ключ в число (индекс), стараясь равномерно распределить ключи по таблице, чтобы уменьшить коллизии.

Понятие коллизии и способы разрешения

- **Коллизия** – случай, когда разные ключи дают одинаковое значение хеш-функции.
- **Способы разрешения**:
    1. Открытая адресация (линейное/квадратичное пробирование, двойное хеширование).
    2. Цепочки (каждая ячейка хранит список коллизий).

Способы реализации хеш-таблицы

1. **Открытая адресация** (все элементы хранятся в самой таблице).
2. **Метод цепочек** (в каждой ячейке – указатель на связный список).

Сложность операции поиска в массивах и в файлах с применением хеш-таблицы

- В **лучшем** (и в среднем) случае операции поиска, вставки и удаления ~ O(1)O(1)O(1).
- В **худшем** – может деградировать до O(n)O(n)O(n).

Тема 10. Иерархические структуры данных

Определение иерархической структуры данных. Основные понятия

- Иерархическая (древовидная) структура – элементы связаны отношениями «родитель-потомок».

Понятие «сильно ветвящегося дерева»

- Дерево, у узлов которого может быть много потомков (не только двое).

Виды бинарных деревьев

1. **Идеально сбалансированное** – все уровни, кроме последнего, полностью заполнены, а в последнем уровне узлы заполняются слева направо без «пробелов».
2. **Дерево выражений** – внутренние узлы – операции, листья – операнды.
3. **Бинарное дерево поиска (BST)** – для каждого узла все ключи в левом поддереве меньше ключа узла, а в правом – больше (или равны).
4. **Сбалансированное дерево** – поддерживает небольшую разницу в высотах поддеревьев (пример – AVL-дерево).
5. **AVL-дерево** – поддерживается равновесие поддеревьев по высоте (разница высот не > 1).
6. **Красно-черное дерево** – каждый узел имеет цвет (красный или чёрный), баланс достигается через правила перекрашивания и вращений.
7. **Косое дерево** (Splay tree) – самонастраивающееся дерево поиска, часто используемое в протоколах сетевых структур.

Асимптотические оценки сложности операций поиска, вставки, удаления

- В **сбалансированных** деревьях (AVL, красно-черных, 2-3-4-деревьях) поиск, вставка, удаление – O(logn)O(\log n)O(logn).
- В несбалансированном BST в худшем случае (если дерево выродилось) – O(n)O(n)O(n).

Сложность по памяти

- Хранение указателей (2–3 на узел и т. п.), в общем ~ O(n)O(n)O(n).

Алгоритм создания идеально сбалансированного бинарного дерева

- Идея: отсортированный массив -> рекурсивно выбираем середину как корень, левые элементы идут в левое поддерево, правые – в правое.

Полное, совершенное (законченное) дерево

- **Полное** – все уровни, кроме, возможно, последнего, заполнены полностью, а узлы последнего уровня расположены «плотно» слева.
- **Совершенное** (perfect) – все уровни полностью заполнены, у каждого узла либо 2 потомка, либо 0 (листья).

Тема 11. Бинарное дерево поиска

Бинарное дерево поиска

- Свойство BST: для каждого узла все ключи левого поддерева < ключа узла, а все ключи правого поддерева > ключа узла.
- Обеспечивает эффективный поиск (O(logn)O(\log n)O(logn)), если дерево «сбалансировано».

Алгоритм операции удаления ключа из дерева

Три основных случая при удалении узла:

1. **Узел – лист** (нет потомков): просто удаляем ссылку из родителя.
2. **Узел имеет одного потомка**: перекидываем ссылку родителя на единственного потомка.
3. **Узел имеет двух потомков**: ищем **преемника** (минимальный из правого поддерева) или **предшественника** (максимальный из левого поддерева), копируем значение, удаляем преемника/предшественника (который уже будет в одной из первых двух ситуаций).

Методы обхода дерева

- **В глубину**:
    - **Прямой (pre-order)**: корень → левое поддерево → правое поддерево.
    - **Симметричный (in-order)**: левое поддерево → корень → правое поддерево (получаем отсортированную последовательность).
    - **Обратный (post-order)**: левое поддерево → правое поддерево → корень.
- **В ширину (по уровням)** – начиная с корня, слева направо на каждом уровне.

Тема 12. Сбалансированные бинарные деревья поиска

Алгоритм вставки ключа в AVL-дерево

1. Стандартная вставка в BST.
2. После вставки «поднимаемся» вверх по дереву, проверяем баланс (разницу высот поддеревьев).
3. Если |разница| > 1, делаем соответствующие **вращения** (поворот дерева) для ребаланса.

Алгоритмы балансировки AVL-дерева

- **Одно поворот** (левый или правый).
- **Двойной поворот** (лево-правый или право-левый).

Назначение балансировки дерева

- Поддерживать высоту дерева порядка O(logn)O(\log n)O(logn), чтобы операции поиска и т. д. оставались эффективными.

Критерий сбалансированности AVL-дерева

- Разница высот левого и правого поддеревьев любого узла не превышает 1.

Критерий несбалансированности AVL-дерева

- Если в каком-то узле разница высот поддеревьев > 1.

Определение высоты сбалансированного дерева при известном числе узлов

- Для AVL-дерева существует связь с числами Фибоначчи. Примерная формула: N(h)=Fh+2−1, N(h) = F_{h+2} - 1,N(h)=Fh+2−1, где N(h)N(h)N(h) – минимальное число узлов при высоте h.

Определение опорного узла в AVL-дереве

- «Опорный узел» – тот, в котором обнаружено нарушение баланса после вставки/удаления и от которого начинаются вращения.

Дерево Фибоначчи

- AVL-дерево минимального размера при заданной высоте h, в котором число узлов соответствует числу Фибоначчи.

Определение количества узлов при заданной высоте дерева

- Для AVL: нижняя и верхняя оценки через числа Фибоначчи.
- Для совершенного дерева высотой h: 2h+1−12^{h+1} - 12h+1−1 узлов.

Совершенное дерево и дерево Фибоначчи. Их сходство и различие

- Оба очень «плотные», но совершенное дерево означает все уровни полностью заполнены, а дерево Фибоначчи – это «минимальное AVL-дерево», где баланс по Фибоначчи.

Тема 12 (Файлы)

Файл как структура данных

- **Файл** – именованная область памяти на внешнем носителе. Можно рассматривать как структуру данных, где элементы – байты, записи и т. п.

Текстовый файл

- Хранит данные в виде последовательности символов (ASCII, Unicode и т. д.).

Двоичный файл

- Хранит данные в «сыром» машинном виде (битовая/байтовая форма), более экономичен.

Объем памяти под числовое значение 0 в двоичном файле

- Зависит от типа данных (int, float и т. п.). Например, 4 байта для 32-битного int.

Структурированный протокол

- Файл может хранить данные по заранее определённой схеме (протоколе). Пример: двоичный формат со структурированными записями (последовательность структур).

Тема 13. Граф

Граф как структура данных

- **Граф** – множество вершин и набор рёбер (связей) между ними.

Отношения в структуре между элементами

- Если граф **ориентированный**, то каждое ребро имеет направление (из одной вершины в другую).
- Если **неориентированный**, то связь «двухсторонняя».

Основные понятия

1. **Простой путь** – последовательность вершин без повторения.
2. **Элементарный путь** – часто синоним простого пути.
3. **Цикл** – путь, у которого начальная и конечная вершины совпадают.
4. **Степень узла** – число рёбер, инцидентных узлу (в ориентированном графе различают входящую и исходящую степень).
5. **Взвешенный граф** – каждому ребру приписан «вес» (стоимость, длина и т. п.).

Определение суммы степеней вершин графа

- Сумма степеней всех вершин в **неориентированном** графе = 2 * (число рёбер).
- В ориентированном графе сумма входящих степеней = сумма исходящих = число рёбер.

Структуры для представления графа в памяти

1. **Список смежности** – для каждой вершины хранится список её соседей (в ориентированном графе – список исходящих рёбер).
2. **Матрица смежности** – квадратная матрица n×n, где n – количество вершин. Элемент (i, j) показывает, есть ли ребро между i и j.

Определение алгоритма обхода графа

- Алгоритм, последовательно посещающий вершины графа, чтобы либо найти путь, либо проверить связность, либо собрать какую-то информацию о графе.

Методы обхода графа

1. **В глубину (DFS)** – рекурсивно спускаемся в глубь, пока возможно. Использует стек (неявный рекурсивный или явный).
2. **В ширину (BFS)** – посещаем вершины «слоями». Использует очередь.

Подграф

- Граф, вершины и рёбра которого являются подмножеством вершин и рёбер исходного графа.

Остовное дерево графа

- Подграф, содержащий все вершины исходного графа и являющийся деревом (т. е. связный и без циклов).
- **Количество рёбер** в остовном дереве для n вершин = n - 1.

Минимальное остовное дерево

- Остовное дерево с минимальной суммой весов рёбер (если граф взвешенный).

Алгоритм Прима

- Построение минимального остовного дерева: начинаем с одной вершины, на каждом шаге добавляем ребро с минимальным весом, которое соединяет уже добавленные вершины с одной из оставшихся.

Алгоритм Крускала

- Сортируем рёбра по возрастанию веса, последовательно добавляем ребро, если оно не создаёт цикла.
- Реализуется при помощи структуры «система непересекающихся множеств» (Union-Find).

Сложность алгоритмов по времени и памяти

- Прима: O(mlogn)O(m \log n)O(mlogn), при использовании очереди с приоритетами (m – число рёбер).
- Крускала: O(mlogm)O(m \log m)O(mlogm) (сортировка рёбер), что эквивалентно O(mlogn)O(m \log n)O(mlogn), так как m≤n2m \leq n^2m≤n2.
- По памяти – хранение структуры Union-Find или очереди.

Алгоритм построения матрицы достижимости

- Можно использовать повторные запуски DFS/BFS из каждой вершины (или алгоритм Уоршалла).
- Матрица достижимости для n вершин будет n×n.

Кратчайший путь в графе

- Взвешенный граф: ищем путь, минимизирующий сумму весов.

Алгоритмы поиска кратчайшего пути

1. **Дейкстры** – для неотрицательных весов рёбер. Сложность O((n+m)logn)O((n + m) \log n)O((n+m)logn) (при реализации через кучу/приоритетную очередь).

2. **Флойда-Уоршалла** – для всех пар вершин, сложность O(n3)O(n^3)O(n3).
3. **Дерево решений** – общее описание подхода, реже используется как отдельный алгоритм.

Сложность алгоритма Дейкстры при реализации на матрице и списке смежности

- При матрице смежности часто ~ O(n2)O(n^2)O(n2).
- При списках смежности и приоритетной очереди ~ O((n+m)logn)O((n + m) \log n)O((n+m)logn).

Тема 14. Сжатие данных

Алгоритмы (методы) сжатия данных

Сжатие – уменьшение объёма данных. Различают:

- **Без потерь** (полностью восстанавливаем исходные данные).
- **С потерями** (часть данных теряется, используется для изображений, звука, видео).

Простые алгоритмы сжатия Лемпеля-Зива: LZ77, LZ78, LZW, RLE

1. **LZ77** – скользящее окно, ищем повторяющиеся последовательности.
2. **LZ78** – строим словарь встречающихся подстрок.
3. **LZW** – динамическое добавление паттернов в словарь во время сжатия.
4. **RLE (Run-Length Encoding)** – замена повторяющихся символов на «символ + счётчик».

Коды фиксированной и переменной длины

- **Фиксированной длины**: например, ASCII (1 байт) для каждого символа.
- **Переменной длины**: более частым символам дают короткие коды.

Оптимальное кодовое дерево, префиксные деревья

- **Префиксные деревья** – никакой код не является префиксом другого кода. Это гарантирует однозначное декодирование.
- **Оптимальное кодовое дерево** (код Хаффмана) – минимизирует среднюю длину кода.

Алгоритм Шеннона-Фано

- Рекурсивно делим набор символов на две группы примерно равной суммарной вероятности, кодируем 0 / 1 и т. д.

Алгоритм Хаффмана

1. Строим дерево снизу вверх: берём два наименее вероятных символа, объединяем в узел с суммой вероятностей, и так далее.
2. Итоговое дерево даёт префиксные коды, оптимальные в смысле минимальной средней длины.

Уметь строить оптимальное кодовое дерево из текста

- Подсчитываем частоту символов, применяем алгоритм Хаффмана.

Декодировать текст по заданному оптимальному кодовому дереву

- Идём по битам кода от корня дерева до листа.

Тема 15. Алгоритмические стратегии

Алгоритмические стратегии

1. **Перебор (brute force)**.
2. **Жадные алгоритмы (greedy)**.
3. **Динамическое программирование (DP)**.
4. **Метод ветвей и границ**.
5. **Разделяй и властвуй** (divide and conquer).
6. И др.

Переборные задачи и их вычислительная сложность

- Перебираем все возможные варианты (комбинации).
- Обычно экспоненциальная сложность.

Постановка задачи перебора при выборе оптимального решения

- Например, задача коммивояжёра (TSP). Нужно проверить все маршруты (перестановки) и выбрать минимальный. Сложность factorial (n!).

Линейный поиск и бинарный поиск в массиве и их стратегии

- **Линейный поиск** – перебираем все элементы (брютфорс по массиву).
- **Бинарный** – «разделяй и властвуй» (делим массив пополам).

Динамическое программирование, основные этапы его применения

1. Разделяем задачу на подзадачи.
2. Находим рекуррентную формулу.
3. Сохраняем результаты подзадач (мемоизация или табличный метод).
4. Используем сохранённые результаты при расчёте более крупных подзадач.

Динамическое программирование – это подход к построению алгоритма, основанный на хранении (кэшировании) результатов промежуточных вычислений с целью избежать повторных вычислений.

Метод ветвей и границ

- Отсекаем ветви поиска, которые заведомо не могут привести к оптимальному решению (сравнивая с текущей лучшей границей).

Жадный алгоритм (greedy)

- На каждом шаге выбираем локально оптимальное решение с надеждой, что оно приведёт к глобально оптимальному. Пример – алгоритм Дейкстры (для неотрицательных рёбер), алгоритм Крускала.

Какие алгоритмические стратегии использует алгоритм Дейкстры

- **Жадная** стратегия (на каждом шаге выбираем вершину с минимальной оценкой расстояния).

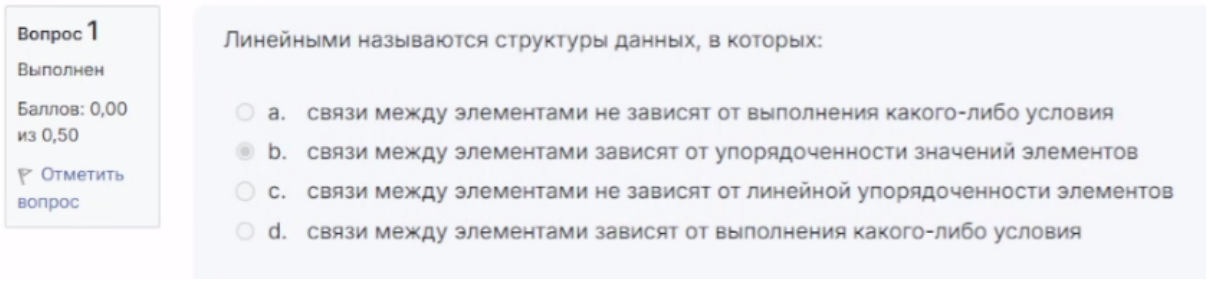
Какую стратегию использует алгоритм Крускала при построении оптимального остовного дерева

- **Жадная** стратегия (на каждом шаге выбираем ребро минимального веса, которое не образует цикла).

Ответы на тест

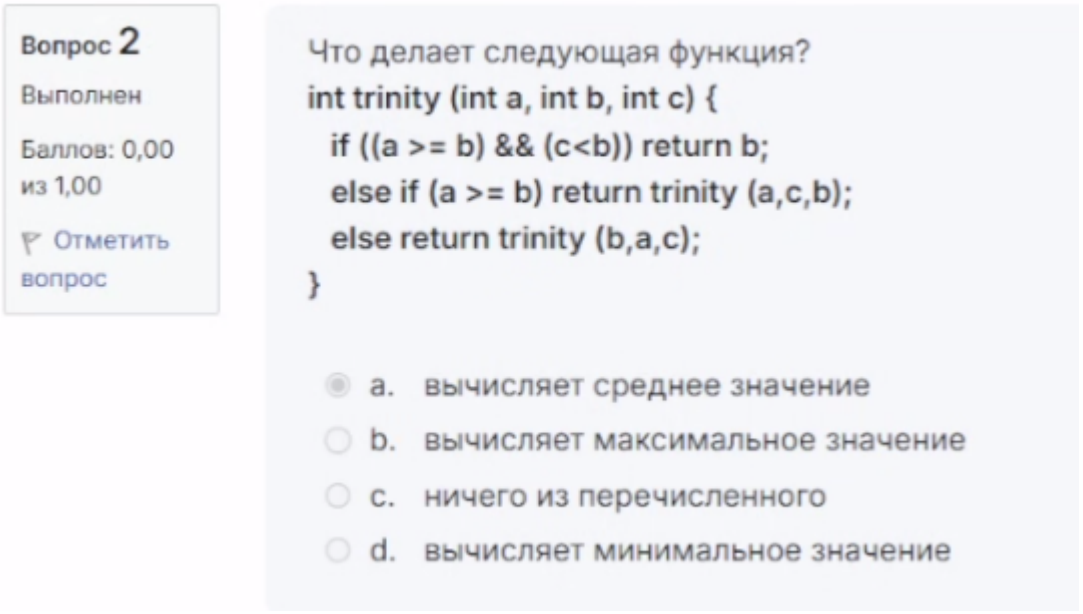
Если ниже не нашли ответ, тут есть все остальные - https://cdn2.mirea.ninja/original/2X/8/8fde944455fcbcaa5a5efcad290e6f4f5567ee14.pdf

Вопрос №1:
Линейными называются структуры данных, в которых:



Правильный ответ: **a. связи между элементами не зависят от выполнения какого-либо условия**

Вопрос №2:
Что делает следующая функция?



Правильный ответ: **Если а неправильно, то с. ничего из перечисленного**

Вопрос №3:
Структура данных, работа с элементами которой организована по принципу FIFO (первый пришел - первый ушел), - это:

Вопрос 3
Выполнен
Баллов: 0,50 из 0,50
Отметить вопрос

Структура данных, работа с элементами которой организована по принципу FIFO (первый пришел - первый ушел), - это:

a. стек
b. дек
c. очередь
d. массив

Правильный ответ: **c. очередь**

Вопрос №4:
Как называется определённая последовательность вычислительных шагов, преобразующих входные величины в выходные?

Вопрос 4
Выполнен
Баллов: 0,50 из 0,50
Отметить вопрос

Как называется определённая последовательность вычислительных шагов, преобразующих входные величины в выходные?

a. Множество данных
b. Структура данных
c. Множество допустимых операций над данными
d. Алгоритм

Правильный ответ: **d. Алгоритм**

Вопрос №5:
Какая вычислительная сложность характерна для алгоритмов со стратегией "разделяй и властвуй" (divide and conquer approach)?

Вопрос 5
Выполнен
Баллов: 0,50 из 0,50
Отметить вопрос

Какая вычислительная сложность характерна для алгоритмов со стратегией "разделяй и властвуй" (divide and conquer approach)?

a. O(n logn)
b. O(1)
c. O(n)
d. O(n^2)

Правильный ответ: **a. O(n logn)**

Вопрос №6:
В теории вычислимости важную роль играет функция Аккермана A(m, n), определённая следующим образом:

Вопрос 6
Выполнен
Баллов: 1,00 из 1,00
Отметить вопрос

В теории вычислимости важную роль играет функция Аккермана A(m, n), определённая следующим образом:

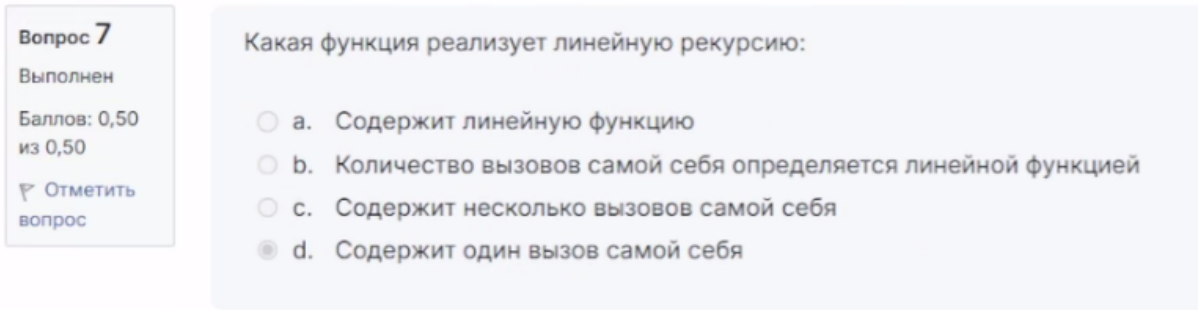
$$A(m,n) = \begin{cases} n+1 & m = 0 \\ A(m-1,1) & m > 0, n = 0 \\ A(m-1, A(m,n-1)) & m > 0, n > 0 \end{cases}$$

Вычислите значение A(2, 2). (Введите только число)

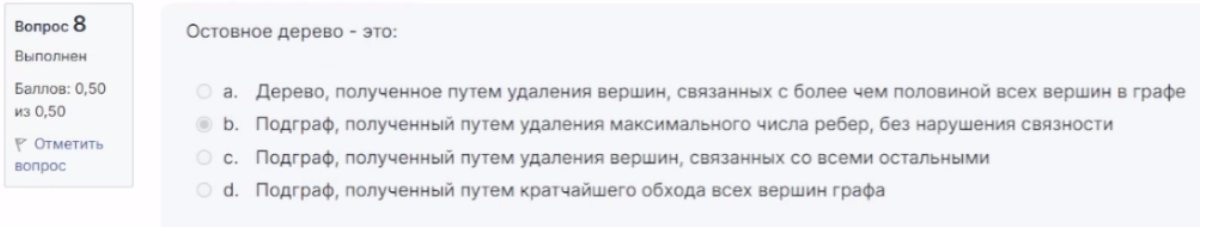
Ответ: 7

Правильный ответ: **7**

Вопрос №7:
Какая функция реализует линейную рекурсию:



Правильный ответ: **d. Содержит один вызов самой себя**
Вопрос №8:
Остовное дерево - это:



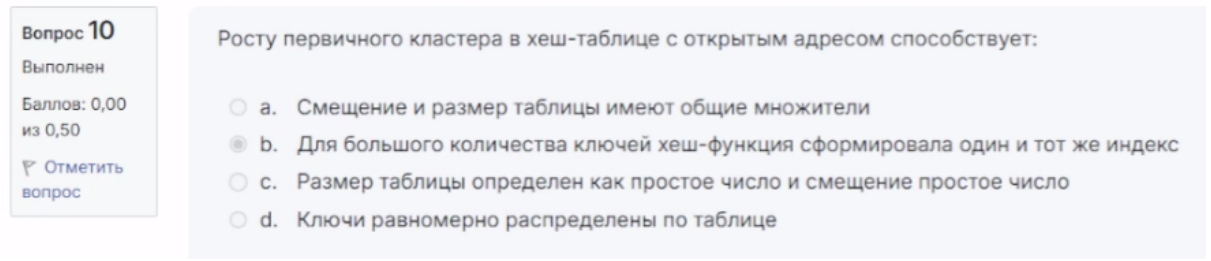
Правильный ответ: **b. Подграф, полученный путем удаления максимального числа ребер, без нарушения связности**

Вопрос №9:
Какое из перечисленных АВЛ-деревьев требует балансировки?

Правильный ответ: **d. 3**

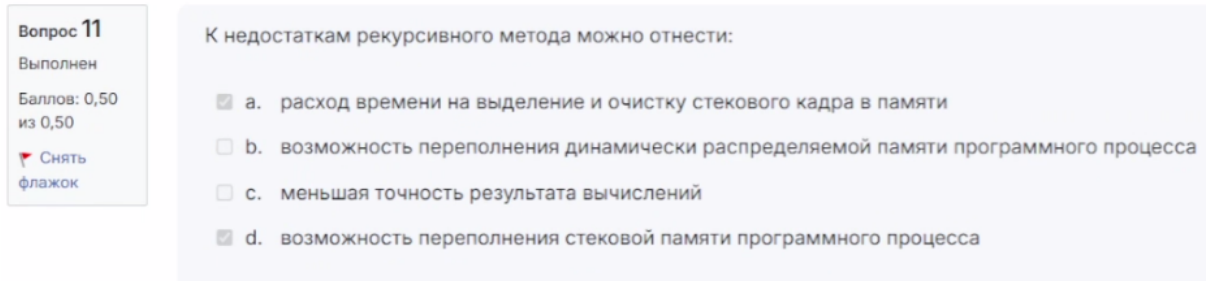
Вопрос №10: ???

Росту первичного кластера в хеш-таблице с открытым адресом способствует:



Правильный ответ: **a. Смещение и размер таблицы имеют общие множители**

Вопрос №11:
К недостаткам рекурсивного метода можно отнести:

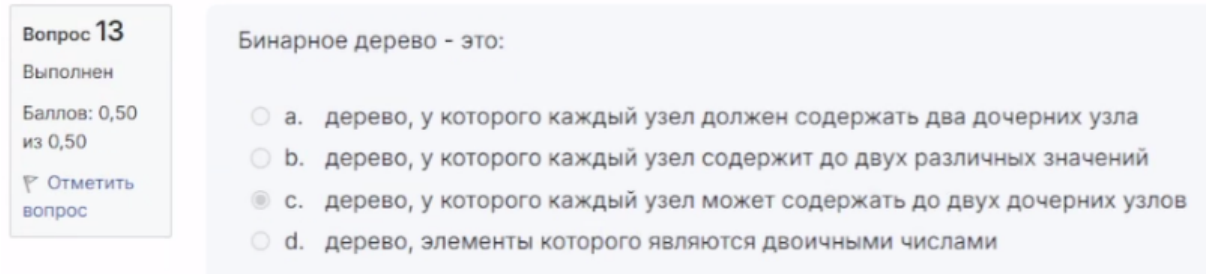


Правильный ответ: **a. расход времени на выделение и очистку стекового кадра в памяти; d. возможность переполнения стековой памяти программного процесса**

Вопрос №12:
Как называется алгоритм, который выполнит сортировку исходного массива (3,1,5,2,4) следующей последовательностью проходов (1,3,5,2,4), (1,3,5,2,4), (1,2,3,5,4), (1,2,3,4,5)

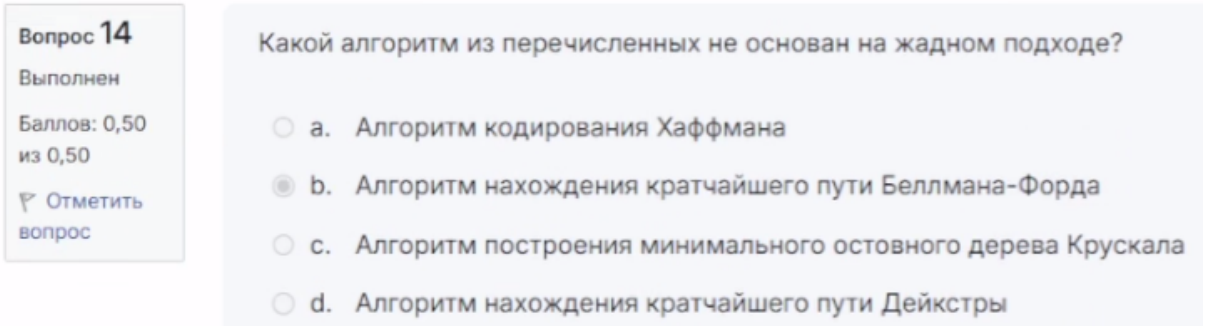
Правильный ответ: **b. Простой вставки**

Вопрос №13:
Бинарное дерево - это:



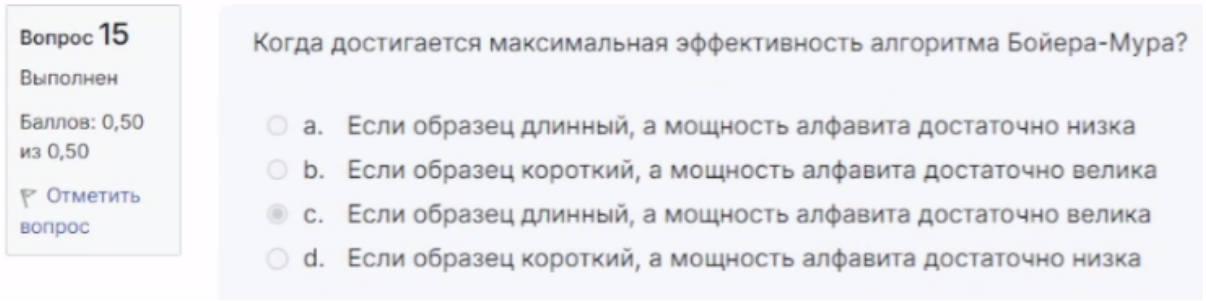
Правильный ответ: **c. дерево, у которого каждый узел может содержать до двух дочерних узлов**

Вопрос №14:
Какой алгоритм из перечисленных не основан на жадном подходе?



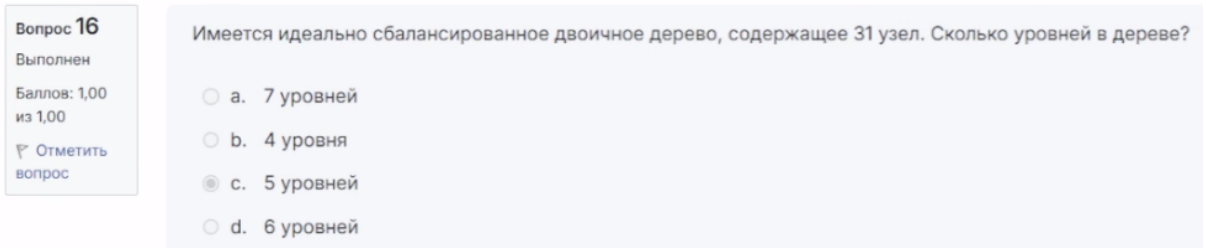
Правильный ответ: **b. Алгоритм нахождения кратчайшего пути Беллмана-Форда**

Вопрос №15:
Когда достигается максимальная эффективность алгоритма Бойера-Мура?



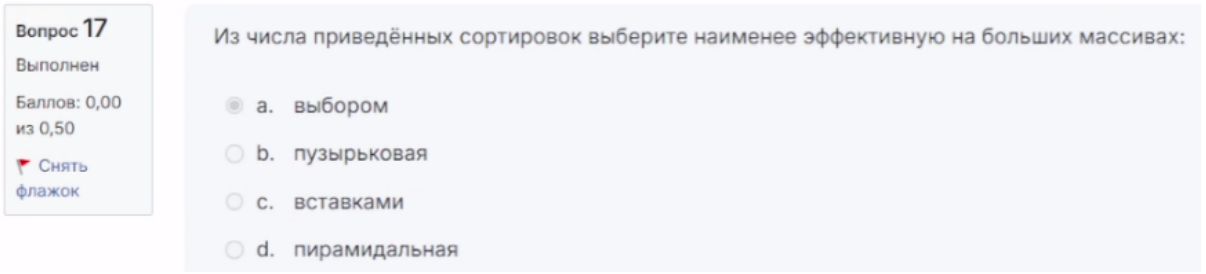
Правильный ответ: **c. Если образец длинный, а мощность алфавита достаточно велика**

Вопрос №16:
Имеется идеально сбалансированное двоичное дерево, содержащее 31 узел. Сколько уровней в дереве?



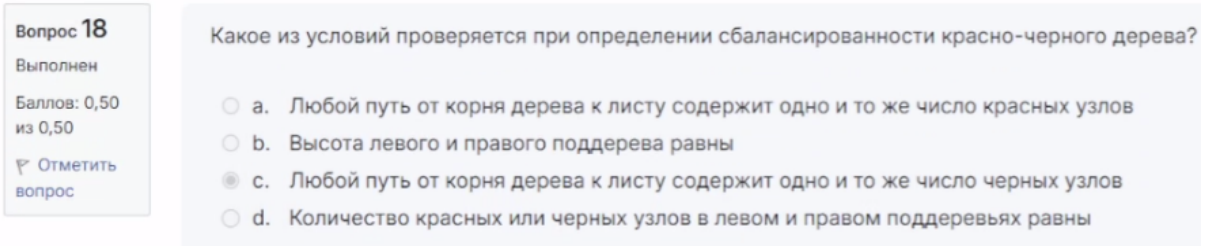
Правильный ответ: **c. 5 уровней**

Вопрос №17:
Из числа приведённых сортировок выберите наименее эффективную на больших массивах:



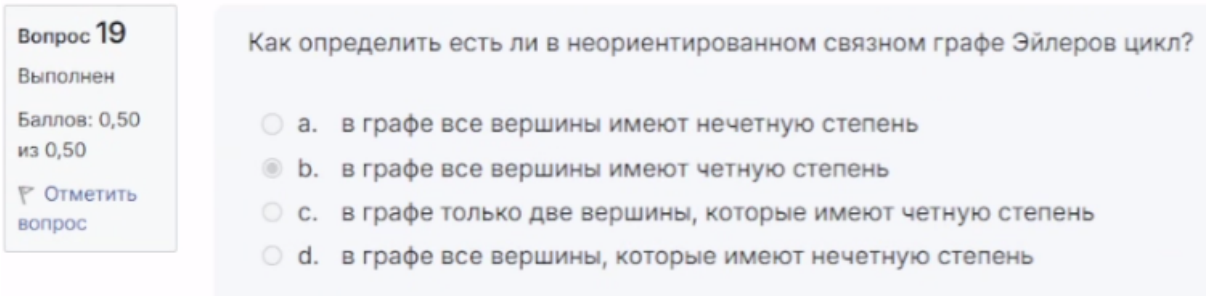
Правильный ответ: **b. пузырьковая**

Вопрос №18:
Какое из условий проверяется при определении сбалансированности красно-черного дерева?



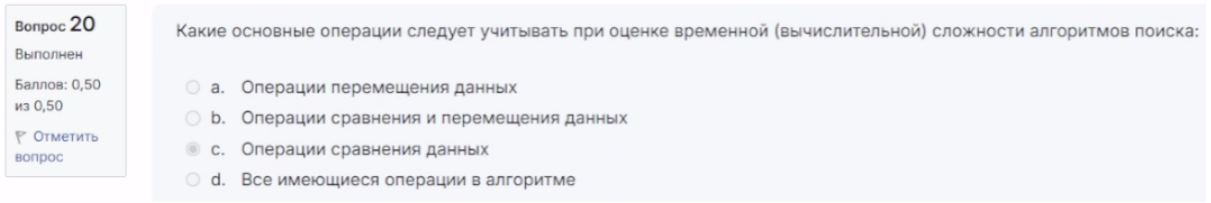
Правильный ответ: **c. Любой путь от корня дерева к листу содержит одно и то же число черных узлов**

Вопрос №19:
Как определить есть ли в неориентированном связном графе Эйлеров цикл?



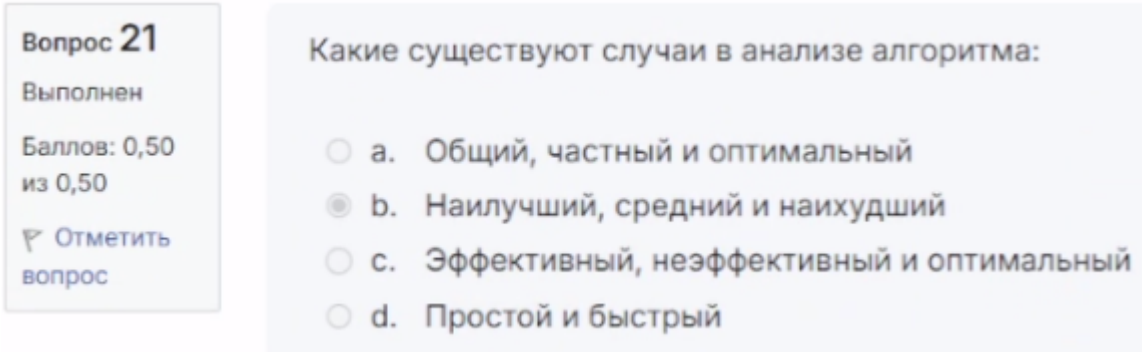
Правильный ответ: **b. в графе все вершины имеют четную степень**

Вопрос №20:
Какие основные операции следует учитывать при оценке временной (вычислительной) сложности алгоритмов поиска:



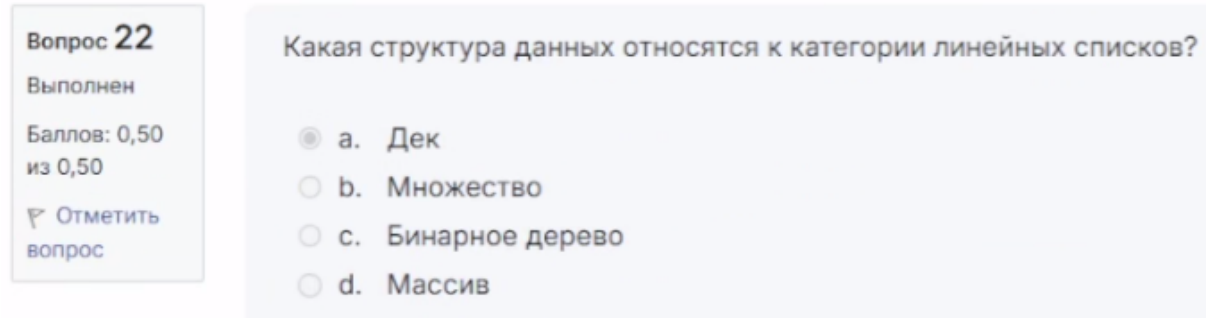
Правильный ответ: **c. Операции сравнения данных**

Вопрос №21:
Какие существуют случаи в анализе алгоритма:



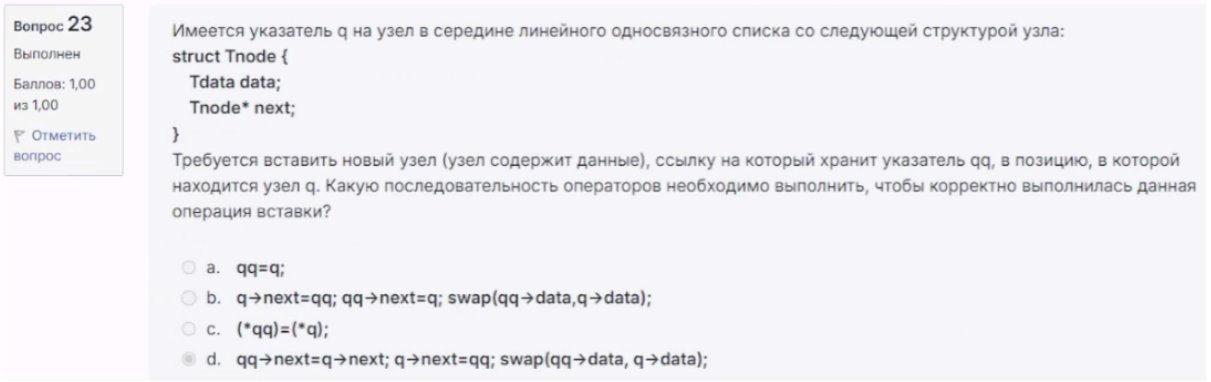
Правильный ответ: **b. Наилучший, средний и наихудший**

Вопрос №22:
Какая структура данных относятся к категории линейных списков?



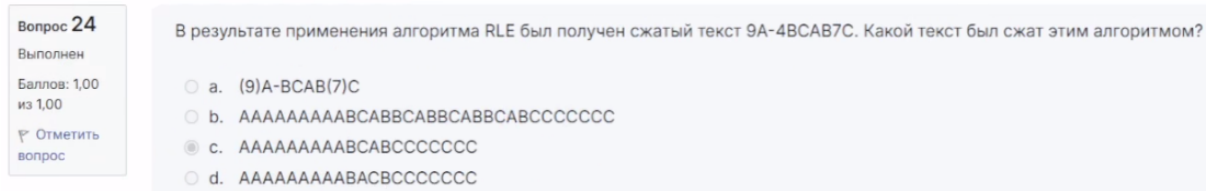
Правильный ответ: **a. Дек**

Вопрос №23:
Имеется указатель q на узел в середине линейного односвязного списка со следующей структурой узла:



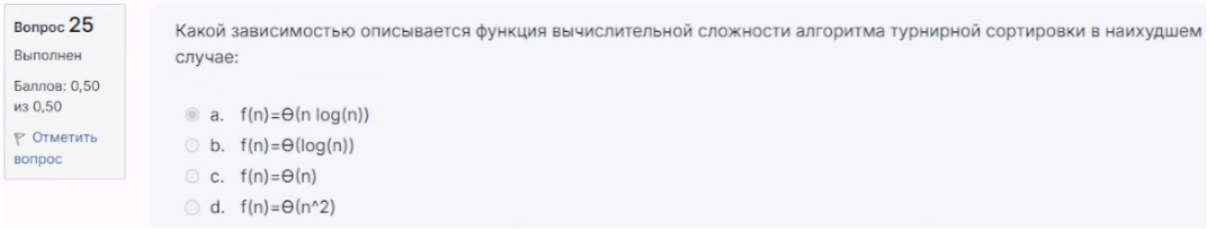
Правильный ответ: **d. qq->next=q->next; q->>next=qq; swap(qq->data, q->data);**

Вопрос №24:
В результате применения алгоритма RLE был получен сжатый текст 9A-4BCAB7C. Какой текст был сжат этим алгоритмом?



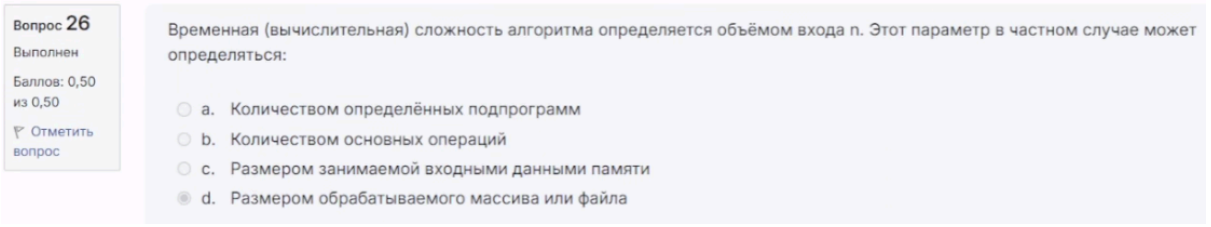
Правильный ответ: **c. AAAAAAAAABCABCCCCCCC**

Вопрос №25:
Какой зависимостью описывается функция вычислительной сложности алгоритма турнирной сортировки в наихудшем случае:



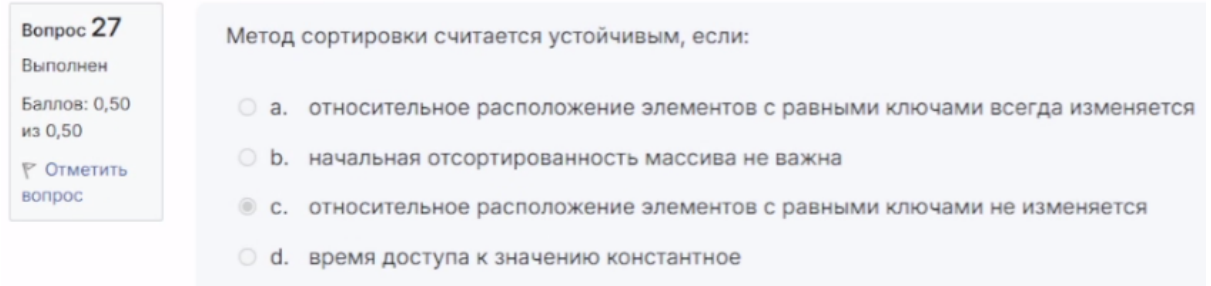
Правильный ответ: **a. f(n)=O(n log(n))**

Вопрос №26:
Временная (вычислительная) сложность алгоритма определяется объёмом входа n. Этот параметр в частном случае может определяться:



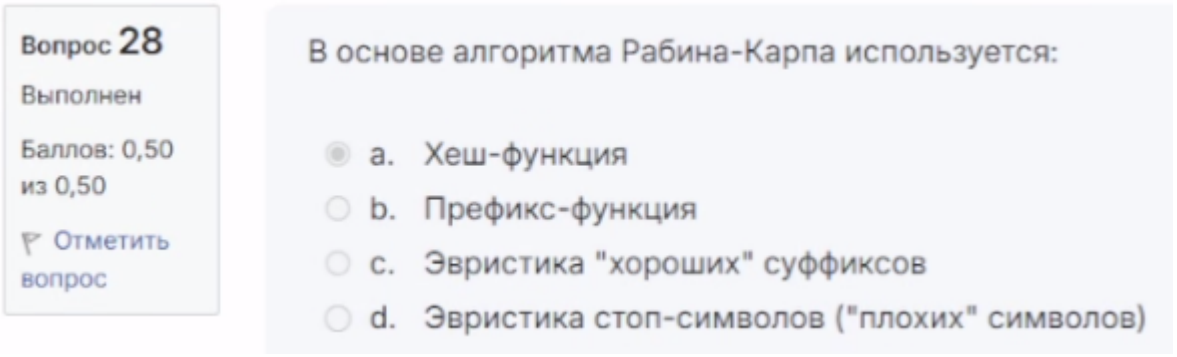
Правильный ответ: **d. Размером обрабатываемого массива или файла**

Вопрос №27:
Метод сортировки считается устойчивым, если:



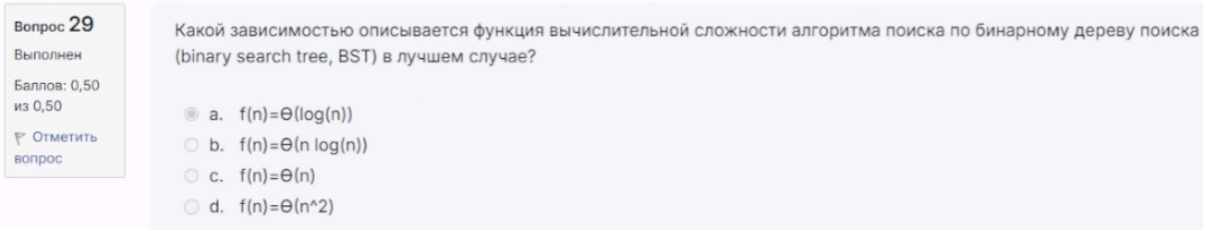
Правильный ответ: **c. относительное расположение элементов с равными ключами не изменяется**

Вопрос №28:
В основе алгоритма Рабина-Карпа используется:



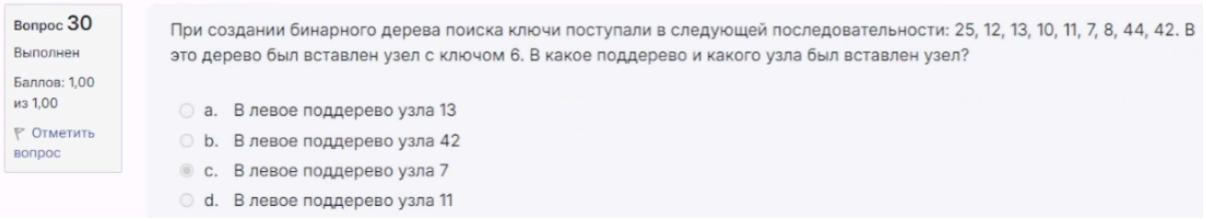
Правильный ответ: **a. Хеш-функция**

Вопрос №29:
Какой зависимостью описывается функция вычислительной сложности алгоритма поиска по бинарному дереву поиска (binary search tree, BST) в лучшем случае?



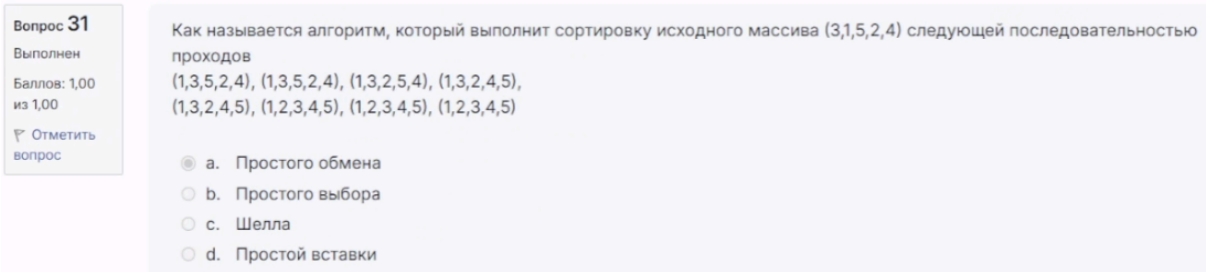
Правильный ответ: **a. f(n)=O(log(n))**

Вопрос №30:
При создании бинарного дерева поиска ключи поступали в следующей последовательности: 25, 12, 13, 10, 11, 7, 8, 44, 42. В это дерево был вставлен узел с ключом 6. В какое поддерево и какого узла был вставлен узел?



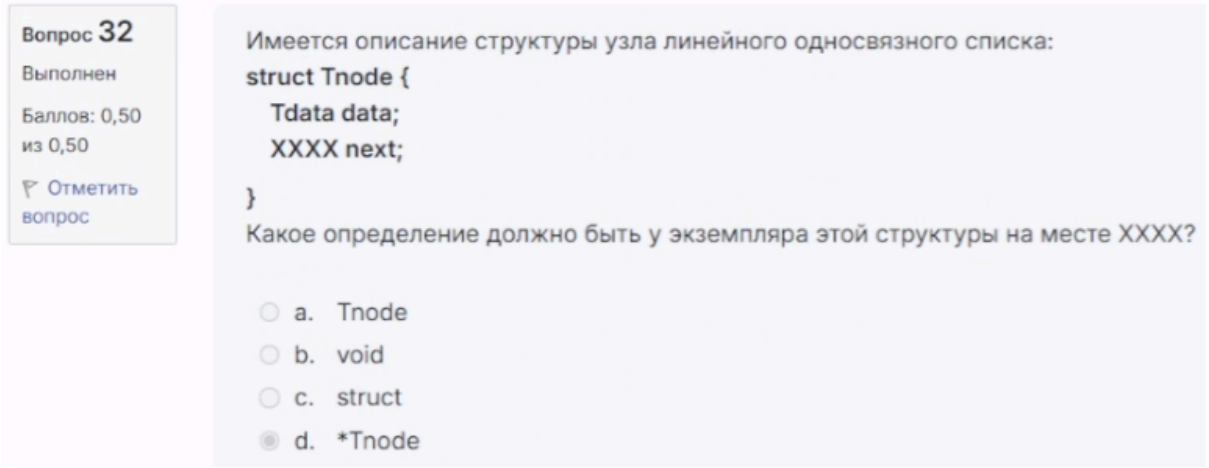
Правильный ответ: **c. В левое поддерево узла 7**

Вопрос №31:
Как называется алгоритм, который выполнит сортировку исходного массива (3,1,5,2,4) следующей последовательностью проходов (1,3,5,2,4), (1,3,5,2,4), (1,3,2,5,4), (1,3,2,4,5), (1,3,2,4,5), (1,2,3,4,5), (1,2,3,4,5), (1,2,3,4,5)



Правильный ответ: **a. Простого обмена**

Вопрос №32:
Имеется описание структуры узла линейного односвязного списка:



Правильный ответ: **d. *Tnode**