ГОУ ВПО Российско-Армянский (Славянский) университет

Утверждено
Директор Института математики и информатики
Арамян Р.Г.
«21» марта 2025, протокол №9.1

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ДИСЦИПЛИНЫ

Наименование дисциплины: Структуры данных и ООП

Автор: к.ф.м.н.Асланян Айк Каренович

Направление подготовки: 01.03.02 Прикладная математика и информатика
Наименование образовательной программы: 01.03.02 Прикладная математика и информатика

1. АННОТАЦИЯ

1.1. Краткое описание содержания данной дисциплины;

Данный курс представляет фундаментальные принципы организации структур данных. Обучающиеся изучат оптимальные методы систематизации, сохранения и манипулирования информацией. Программа охватывает исследование линейных и нелинейных форм организации данных, включая соответствующие им алгоритмические операции.

1.2. Трудоемкость в академических кредитах и часах, формы итогового контроля (экзамен/зачет);

Виды учебной работы	**В акад. часах**
Общее количество часов	96
Лекция	32
Практика	64
Итоговый контроль	Экзамен

1.3. Взаимосвязь дисциплины с другими дисциплинами учебного плана специальности (направления)

Дисциплина занимает центральное место в системе подготовки специалистов и тесно интегрирована с другими дисциплинами учебного плана. Она опирается на фундаментальные знания, полученные в курсах «Алгоритмы и алгоритмические языки (язык С)» и «Дискретная математика». Одновременно курс служит основой для последующего изучения курсов "Алгоритмы" и "Баз данных", предоставляя необходимый концептуальный аппарат для понимания принципов организации и обработки информации. Таким образом, дисциплина формирует связующее звено между теоретическими основами информатики и их практической реализацией в профессиональной деятельности.

1.4. Результаты освоения программы дисциплины:

Код компетенции	**Наименование компетенции**	**Код индикатора достижения компетенций**	**Наименование индикатора достижений компетенций**
ПК-7	способностью к разработке и применению алгоритмических и	1	Знать методы и технологии разработки и применения

	программных решений в области системного и прикладного программного обеспечения		системного и прикладного программного обеспечения
		2	Разрабатывать и применять алгоритмические и программные решения в области системного и прикладного программного обеспечения
		3	Владеть способностью разрабатывать и применять алгоритмические и программные решения в области системного и прикладного программного обеспечения
ОПК-5	Способен разрабатывать алгоритмы и компьютерные программы, пригодные для практического применения	1	Обладает знаниями в области алгоритмизации и программирования
		2	Демонстрирует умение выбрать и обосновать выбор языка и среды программирования для разработки компьютерных программ
		3	Владеет навыками разработки алгоритмов и компьютерных программ, пригодных для практического применения

2. УЧЕБНАЯ ПРОГРАММА

2.1. Цели и задачи дисциплины

Целью дисциплины является формирование у студентов системных знаний о принципах организации, хранения и обработки информации в компьютерных системах, а также развитие навыков эффективного проектирования алгоритмов. Основными задачами курса выступают изучение фундаментальных типов структур данных и алгоритмов их обработки, освоение методов анализа временной и пространственной сложности, формирование умений выбора оптимальных структур данных для решения конкретных прикладных задач, а также развитие способности к критическому анализу эффективности различных подходов к организации информации.

2.2. Трудоемкость дисциплины и виды учебной работы (в академических часах и зачетных единицах)

Виды учебной работы	**Всего, в акад. часах**	**Распределение по семестрам**					
		3 сем	**___ сем**	**___ сем**	**___ сем.**	**___ сем**	**____ сем.**
1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1. Общая трудоемкость изучения дисциплины по семестрам, в т. ч.:	**180**	**180**					
1.1. Аудиторные занятия, в т. ч.:	**96**	**96**					
1.1.1. Лекции	**32**	**32**					
1.1.2. Практические занятия, в т. ч.	**64**	**64**					
1.1.2.1. Обсуждение прикладных проектов	**64**	**64**					
1.2. Самостоятельная работа, в т. ч.:	**57**	**57**					
1.3. Другие методы и формы занятий	**27**	**27**					
Итоговый контроль (Экзамен, Зачет, диф. зачет - указать)		**Экзамен**					

2.3. Содержание дисциплины

2.3.1. Тематический план и трудоемкость аудиторных занятий (модули, разделы дисциплины и виды занятий) по рабочему учебному плану

Разделы и темы дисциплины	**Всего (ак. часов)**	**Лекции(ак. часов)**	**Практ. Занятия (ак. часов)**	**Семинары (ак. часов)**	**Лабор. (ак. часов)**
1	**2=3+4+5+6+7**	**3**	**4**	**5**	**6**

Тема 1. Структуры данных, введение (назначение, сложность алгоритмов, O,Θ,Ω). STL.	**6**	**2**	**4**	**0**	**0**
Тема 2. Структура данных вектор. Оценка сложности операций, циклов, рекурсивных функций.	**6**	**2**	**4**	**0**	**0**
Тема 3. Односвязные списки.	**6**	**2**	**4**	**0**	**0**
Тема 4. Двусвязные списки.	**6**	**2**	**4**	**0**	**0**
Тема 5. Структура данных очередь, двусторонняя очередь.	**6**	**2**	**4**	**0**	**0**
Тема 6. Структура данных стек.	**6**	**2**	**4**	**0**	**0**
Тема 7. Бинарные деревья, обходы.	**6**	**2**	**4**	**0**	**0**
Тема 8. Бинарное дерево поиска.	**6**	**2**	**4**	**0**	**0**
Тема 9. Сбалансированные деревья, красно-черные деревья.	**12**	**4**	**8**	**0**	**0**
Тема 10. Пирамида, пирамидальная сортировка, очередь с приоритетами.	**12**	**4**	**8**	**0**	**0**
Тема 11. Хэш-таблицы: прямая адресация, коллизии.	**12**	**4**	**8**	**0**	**0**
Тема 12. Хэш-таблицы: открытая адресация.	**12**	**4**	**8**	**0**	**0**
ИТОГО	**96**	**32**	**64**	**0**	**0**

2.3.2. Краткое содержание разделов дисциплины в виде тематического плана

Тема 1. Структуры данных, введение (назначение, сложность алгоритмов, O,Θ,Ω). STL.

Структуры данных представляют собой способы организации и хранения информации в памяти компьютера для обеспечения эффективного доступа и модификации данных. Для анализа их производительности используется асимптотическая нотация: O (Big O) описывает верхнюю границу сложности в наихудшем случае, Θ (Theta) дает точную асимптотическую оценку когда верхняя и нижняя границы совпадают, а Ω (Omega) определяет нижнюю границу в наилучшем случае. STL (Standard Template Library) в C++ предоставляет готовые реализации основных структур данных через контейнеры (vector, list, map, set), итераторы для навигации и алгоритмы для обработки данных, обеспечивая типобезопасность и высокую производительность при разработке программного обеспечения.

Тема 2. Структура данных вектор. Оценка сложности операций, циклов, рекурсивных функций.

Вектор — это динамический массив, который автоматически изменяет размер при необходимости. Доступ к элементам по индексу выполняется за O(1), добавление в конец имеет амортизированную сложность O(1), а вставка/удаление в произвольной позиции — O(n).

Асимптотическая сложность циклов зависит от количества итераций: простой цикл дает O(n), вложенные циклы — $O(n^2)$, циклы с логарифмическим изменением — O(log n). Рекурсивные функции анализируются через рекуррентные соотношения: бинарный поиск имеет сложность O(log n) благодаря делению задачи пополам, а наивное вычисление чисел Фибоначчи — O(2^n) из-за повторных вычислений.

Тема 3. Односвязные списки.

Односвязный список — это линейная структура данных, состоящая из узлов, каждый из которых содержит данные и указатель на следующий элемент. Доступ к элементам осуществляется только последовательно от головы списка, что дает сложность O(n) для поиска элемента по значению или позиции. Вставка и удаление элементов в начале списка выполняются за O(1), поскольку требуют только изменения указателей, однако вставка/удаление в произвольной позиции имеет сложность O(n) из-за необходимости последовательного прохода до нужного места. Основные преимущества односвязных списков — динамическое управление памятью и эффективные операции с началом списка, но они уступают массивам в скорости произвольного доступа и требуют дополнительной памяти для хранения указателей.

Тема 4. Двусвязные списки.

Двусвязный список — это линейная структура данных, где каждый узел содержит данные и два указателя: на следующий и предыдущий элементы. Доступ к элементам остается последовательным с сложностью O(n) для поиска по значению или позиции, но появляется возможность эффективного обхода в обоих направлениях. Вставка и удаление элементов в начале или конце списка выполняются за O(1), поскольку не требуют полного прохода структуры благодаря наличию указателей в обе стороны. Удаление узла по известному указателю также происходит за O(1), что является преимуществом перед односвязными списками, где необходимо найти предыдущий элемент. Двусвязные списки обеспечивают более гибкую навигацию и эффективные операции с концами структуры, но требуют дополнительной памяти для хранения обратных указателей и усложняют логику поддержания целостности связей.

Тема 5. Структура данных очередь, двусторонняя очередь.

Очередь — это линейная структура данных, работающая по принципу FIFO (First In, First Out), где элементы добавляются в конец (enqueue) и удаляются из начала (dequeue). Основные операции выполняются за O(1): добавление элемента в хвост, удаление из головы, а также доступ к первому элементу без удаления. Поиск произвольного элемента требует O(n) времени, поскольку доступ возможен только через голову очереди. Двусторонняя очередь (deque) расширяет функциональность обычной очереди, позволяя эффективно добавлять и удалять элементы с обеих сторон за O(1). В deque доступны операции push_front, push_back, pop_front и pop_back, что делает её универсальной структурой, способной работать как очередь, стек или комбинация обеих. Обе структуры широко используются в

алгоритмах обхода графов, планировании задач и буферизации данных, обеспечивая предсказуемый порядок обработки элементов.

Тема 6. Структура данных стек

Стек — это линейная структура данных, работающая по принципу LIFO (Last In, First Out), где элементы добавляются и удаляются только с одного конца, называемого вершиной стека. Основные операции выполняются за O(1): push (добавление элемента на вершину), pop (удаление элемента с вершины) и top/peek (доступ к верхнему элементу без удаления). Поиск произвольного элемента требует O(n) времени, поскольку доступ возможен только через вершину стека с последовательным удалением элементов. Стек естественным образом моделирует рекурсивные вызовы функций, где каждый новый вызов помещается на вершину, а при завершении функции её контекст удаляется. Структура широко применяется для проверки сбалансированности скобок, вычисления выражений в постфиксной нотации, реализации алгоритмов обхода графов в глубину (DFS) и отмены операций (undo), обеспечивая эффективное управление данными с обратным порядком обработки.

Тема 7. Бинарные деревья, обходы

Бинарное дерево — это иерархическая структура данных, где каждый узел имеет не более двух потомков (левый и правый), образуя древовидную структуру с корневым узлом на вершине. Поиск, вставка и удаление элементов в сбалансированном бинарном дереве поиска выполняются за O(log n) благодаря делению пространства поиска пополам на каждом уровне, однако в вырожденном случае (когда дерево превращается в линейную цепочку) сложность достигает O(n). Существуют три основных способа обхода бинарного дерева: прямой обход (preorder) посещает узлы в порядке корень-левый-правый, симметричный обход (inorder) дает последовательность левый-корень-правый и для дерева поиска возвращает элементы в отсортированном порядке, а обратный обход (postorder) следует схеме левый-правый-корень. Все виды обхода имеют временную сложность O(n), поскольку каждый узел посещается ровно один раз, и могут быть реализованы как рекурсивно, так и итерационно с использованием стека для хранения промежуточных состояний.

Тема 8. Бинарное дерево поиска

Бинарное дерево поиска (BST) — это специализированная структура данных, где для каждого узла все элементы в левом поддереве меньше значения узла, а все элементы в правом поддереве больше. Эта упорядоченность обеспечивает эффективные операции поиска, вставки и удаления со сложностью O(log n) в среднем случае для сбалансированного дерева, поскольку на каждом шаге исключается половина оставшихся узлов. Однако в худшем случае, когда дерево вырождается в линейную структуру (например, при вставке элементов в отсортированном порядке), сложность достигает O(n). Поиск элемента начинается с корня и рекурсивно спускается влево или вправо в зависимости от сравнения искомого значения с текущим узлом. Вставка следует аналогичному алгоритму до достижения листового узла, а удаление требует рассмотрения трех случаев: удаление листа, узла с одним потомком и узла с двумя потомками, где применяется замена на наименьший элемент правого поддерева или наибольший элемент левого поддерева.

Тема 9. Сбалансированные деревья, красно-черные деревья

Сбалансированные деревья — это самокорректирующиеся структуры данных, которые автоматически поддерживают приблизительно равную высоту левого и правого поддеревьев, гарантируя логарифмическую сложность O(log n) для операций поиска, вставки и удаления даже в худшем случае. Красно-черное дерево является одним из наиболее распространенных типов сбалансированных деревьев, где каждый узел окрашен в красный или черный цвет

согласно строгим правилам: корень всегда черный, красные узлы не могут иметь красных потомков, все пути от корня до листьев содержат одинаковое количество черных узлов. Эти ограничения гарантируют, что самый длинный путь не более чем в два раза длиннее самого короткого, обеспечивая высоту дерева не более $2 \cdot \log(n+1)$. При вставке и удалении элементов выполняются операции перекраски узлов и поворотов (левый/правый поворот) для восстановления баланса, что происходит за O(log n) времени. Красно-черные деревья широко используются в системных библиотеках (например, в std::map и std::set в C++), поскольку обеспечивают стабильную производительность и относительную простоту реализации по сравнению с другими сбалансированными структурами.

Тема 10. Пирамида, пирамидальная сортировка, очередь с приоритетами

Пирамида (heap) — это полное бинарное дерево, удовлетворяющее свойству порядка: в максимальной пирамиде каждый родительский узел больше своих потомков, а в минимальной — меньше. Основные операции выполняются за O(log n): вставка элемента требует "всплытия" нового узла вверх по дереву, а удаление корня (максимального/минимального элемента) — "просеивания" вниз замещающего элемента. Построение пирамиды из n элементов происходит за O(n) благодаря алгоритму heapify, который обрабатывает узлы снизу вверх. Пирамидальная сортировка (heapsort) использует свойства пирамиды для сортировки массива: сначала строится максимальная пирамида за O(n), затем n раз извлекается максимальный элемент и помещается в конец массива, что дает общую сложность O(n log n) в любом случае и требует O(1) дополнительной памяти. Очередь с приоритетами естественно реализуется через пирамиду, где элементы обслуживаются не по порядку поступления, а по приоритету: операции вставки и извлечения элемента с наивысшим приоритетом выполняются за O(log n), что делает структуру идеальной для алгоритмов Дейкстры, A* и планирования задач.

Тема 11. Хэш-таблицы: прямая адресация, коллизии

Хэш-таблица — это структура данных, использующая хэш-функцию для преобразования ключей в индексы массива, обеспечивая в среднем O(1) для операций поиска, вставки и удаления. Прямая адресация представляет простейший случай, когда ключи являются небольшими неотрицательными целыми числами и используются непосредственно как индексы массива, что гарантирует O(1) для всех операций без коллизий, но требует память размером равным максимальному возможному ключу. Коллизии возникают когда хэш-функция отображает разные ключи в одну и ту же ячейку таблицы, что неизбежно при ограниченном размере массива. Основные методы разрешения коллизий включают метод цепочек, где коллизирующие элементы хранятся в связанных списках по каждому индексу, и открытую адресацию с линейным зондированием (проверка следующих ячеек по порядку), квадратичным зондированием или двойным хэшированием для поиска свободного места. Эффективность хэш-таблицы зависит от качества хэш-функции и коэффициента заполнения: хорошая функция равномерно распределяет ключи, поддерживая среднюю сложность O(1), но в худшем случае при множественных коллизиях сложность может деградировать до O(n).

Тема 12. Хэш-таблицы: открытая адресация

Открытая адресация — это метод разрешения коллизий в хэш-таблицах, где все элементы хранятся непосредственно в массиве таблицы, а при возникновении коллизии используется последовательность проб для поиска свободной ячейки. Основные стратегии зондирования включают линейное зондирование (проверка ячеек $h(k)$, $h(k)+1$, $h(k)+2$, ...), которое простое в реализации но склонно к образованию кластеров смежных занятых ячеек, квадратичное зондирование ($h(k)$, $h(k)+1^2$, $h(k)+2^2$, ...), которое уменьшает кластеризацию но может не исследовать все ячейки таблицы, и двойное хэширование с использованием второй

хэш-функции для определения шага зондирования. При вставке элемента выполняется поиск первой свободной ячейки по выбранной стратегии, а при удалении необходимо помечать ячейки как "удаленные" вместо простой очистки, чтобы не нарушить цепочки поиска других элементов. Открытая адресация обеспечивает лучшую локальность памяти по сравнению с методом цепочек и избегает накладных расходов на указатели, но требует поддержания коэффициента заполнения ниже критического уровня (обычно 0.5-0.7) для сохранения эффективности операций, иначе производительность резко снижается из-за длинных последовательностей проб.

Литература: Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ = Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Introduction to Algorithms. Second edition / пер. с англ. канд. техн. наук И. В. Красикова, Н. А. Ореховой, В. Н. Романова под ред. канд. техн. наук И. В. Красикова. — 2-е изд. — М.: Издательский дом «Вильямс», 2011. — 1290 с., ил. — 1000 экз. — ISBN 978-5–8459–0857–5 (рус.). — ISBN 0–07–013151–1 (англ.).

2.3.3. Краткое содержание семинарских/практических занятий/лабораторного практикума

«Семинарские/практические занятия по структурам данных» включают программную реализацию основных структур и алгоритмов для закрепления теоретических знаний. Студенты изучают работу с массивами и STL-контейнерами (vector, list, stack, queue, deque), реализуют односвязные и двусвязные списки с операциями вставки, удаления и поиска элементов. Практические задания охватывают создание классов для стека и очереди, программирование рекурсивных алгоритмов обхода бинарных деревьев (preorder, inorder, postorder) и реализацию бинарного дерева поиска с базовыми операциями. Лабораторные работы включают изучение алгоритмов сортировки (quicksort, mergesort, heapsort) с анализом их временной сложности, реализацию пирамиды и очереди с приоритетами, а также создание простых хэш-таблиц с разрешением коллизий методом цепочек и открытой адресацией. Все задания сопровождаются тестированием на различных наборах данных, измерением производительности и сравнением асимптотической сложности с практическими результатами, что развивает навыки анализа эффективности алгоритмов и выбора оптимальных структур данных для конкретных задач.

2.3.4. Материально-техническое обеспечение дисциплины

«Материально-техническое обеспечение дисциплины» включает компьютерные классы с современными персональными компьютерами, оснащенными процессорами не менее 2 ГГц, оперативной памятью от 4 ГБ и достаточным дисковым пространством для установки IDE и компиляторов. Программное обеспечение включает среды разработки (Visual Studio или аналогичные), компиляторы C++ (GCC, Clang, MSVC), отладчики и профилировщики для анализа производительности алгоритмов. Аудитории оборудованы проекторами и интерактивными досками для демонстрации алгоритмов и структур данных. Лабораторные занятия требуют доступа к сети Интернет для работы с онлайн-ресурсами. Дополнительно обеспечивается доступ к электронным учебникам, базам данных с алгоритмическими задачами и специализированным программам для визуализации структур данных, что способствует лучшему пониманию материала и развитию практических навыков программирования.

2.4. Модульная структура дисциплины с распределением весов по формам контролей

Формы контролей	Вес формы (форм) текущего контроля в результирующей оценке текущего контроля (по модулям)		Вес формы промежуточного контроля в итоговой оценке промежуточного контроля		Вес итоговой оценки промежуточного контроля в результирующей оценке промежуточных контролей		Вес итоговой оценки промежуточного контроля в результирующей оценке промежуточных контролей (семестровой оценке)	Веса результирующей оценки промежуточных контролей и оценки итогового контроля в результирующей оценке итогового контроля
Вид учебной работы/контроля	**M1** [1]	**M2**	**M1**	**M2**	**M1**	**M2**		
Контрольная работа	0.5	0.5						
Веса результирующих оценок текущих контролей в итоговых оценках промежуточных контролей					0.4	0.6		
Веса оценок промежуточных контролей в итоговых оценках промежуточных контролей								
Вес итоговой оценки 1-го промежуточного контроля в результирующей оценке промежуточных контролей							0.4	
Вес итоговой оценки 2-го промежуточного контроля в результирующей оценке промежуточных контролей							0.6	
Вес результирующей оценки промежуточных контролей в результирующей оценке итогового контроля								0.4

[1] Учебный Модуль

Вес итогового контроля (Экзамен/зачет) в результирующей оценке итогового контроля								0.6
	∑ = 1	**∑ = 1**	**∑ = 1**	**∑ = 1**	**∑ = 1**	**∑ = 1**	**∑ = 1**	**∑ = 1**

3. Теоретический блок

3.1. Материалы по теоретической части курса

3.1.1. Учебник(и);

Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы: построение и анализ = Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Introduction to Algorithms. Second edition / пер. с англ. канд. техн. наук И. В. Красикова, Н. А. Ореховой, В. Н. Романова под ред. канд. техн. наук И. В. Красикова. — 2-е изд. — М.: Издательский дом «Вильямс», 2011. — 1290 с., ил. — 1000 экз. — ISBN 978-5–8459–0857–5 (рус.). — ISBN 0–07–013151–1 (англ.).

4. Фонды оценочных средств.

4.3.4. Наглядно-иллюстративные материалы
Слайды с деревьями, хеш-таблицами, пирамидой
Визуализаторы: https://visualgo.net, https://csacademy.com/app/graph_editor/
Графические схемы вставки/удаления элементов
4.3.5. Другие виды материалов
GitHub-репозитории с реализациями всех структур
Видео-уроки: канал mycodeschool, курс от MIT (YouTube)
Консольные симуляторы структур данных

4.4. Вопросы и задания для самостоятельной работы студентов
Примеры:
Реализовать стек на массиве и списке. Сравнить производительность.
Написать итератор для собственного списка.
Реализовать кучу (heap) и выполнить пирамидальную сортировку массива.
Сравнить эффективность поиска в BST и в хеш-таблице.
Реализовать очередь с приоритетами на базе вектора и пирамиды.

4.6. Образцы вариантов контрольных работ, тестов и/или других форм текущих и промежуточных контролей
Вариант контрольной работы:
Определите временную сложность вставки в std::vector.
Реализуйте рекурсивный обход бинарного дерева.
Объясните различия между хешированием с цепочками и открытой адресацией.

4.7. Перечень экзаменационных вопросов
Обозначения O, Ω, Θ и их роль в анализе алгоритмов
Структура данных "вектор" и его поведение при перераспределении памяти
Отличия односвязного и двусвязного списка
Принцип работы очереди и дека, реализация с помощью STL
Стек и его применение при рекурсии
Прямой, симметричный и обратный обходы бинарного дерева
Бинарное дерево поиска: реализация, особенности
Принцип работы красно-черного дерева
Приоритетные очереди и их реализация через кучу
Основы хеширования и типы коллизий
Методы разрешения коллизий: цепочки и открытая адресация

4.8. Образцы экзаменационных билетов
Билет №2
Объясните сложность операций в std::vector.
Реализуйте односвязный список и напишите функцию вставки.
Опишите метод открытой адресации и его преимущества.
4.9. Образцы экзаменационных практических заданий
Написать реализацию хеш-таблицы с использованием цепочек
Реализовать пирамидальную сортировку массива
Построить BST и реализовать поиск по значению
Реализовать стек с функцией "получить максимум" за O(1)

4.10. Банк тестовых заданий для самоконтроля
Примеры:
Что верно о std::deque?
а) Реализован как кольцевой буфер
б) Быстрая вставка в начало и конец
в) Позволяет произвольный доступ
г) Все вышеперечисленное
Ответ: г
В каком случае std::vector перераспределяет память?
Ответ: При превышении текущей вместимости (capacity)
Что означает Θ(n)?
Ответ: Функция имеет точную асимптоту n

4.11. Методики решения и ответы к образцам тестовых заданий
Вопрос: Что делает std::unordered_map при коллизии?
Ответ: Использует списки (цепочки) для хранения элементов с одинаковым хешем.
Пояснение: Это форма хеширования с разрешением коллизий через цепочки.
Вопрос: Как реализуется пирамида в STL?
Ответ: Через std::priority_queue, обычно поверх std::vector и make_heap.

5. Методический блок
5.1. Методика преподавания

5.1.1. Методические рекомендации для студентов по подготовке к семинарским, практическим или лабораторным занятиям, по организации самостоятельной работы студентов при изучении конкретной дисциплины.

Методика преподавания дисциплины основывается на сочетании теоретических лекций с интенсивной практической работой, где каждая новая структура данных изучается через последовательность "теория → демонстрация → самостоятельная реализация → анализ сложности". Студентам рекомендуется готовиться к семинарским занятиям путем предварительного изучения теоретического материала, выполнения базовых упражнений на бумаге для понимания алгоритмов, а к лабораторным работам — через настройку среды разработки и повторение синтаксиса языка программирования. Самостоятельная работа включает решение задач различной сложности, чтение дополнительной литературы по алгоритмам, создание собственных тестовых наборов для проверки реализаций и ведение дневника с анализом временной и пространственной сложности решенных задач. Особое внимание уделяется постепенному переходу от использования готовых STL-контейнеров к самостоятельной реализации структур данных, что развивает глубокое понимание их внутреннего устройства и принципов работы, необходимых для эффективного решения алгоритмических задач в профессиональной деятельности.