**Вопросы для самоконтроля**

1. *Сформулируйте общую идею амортизационного анализа.*

Общая идея заключается в том, что любая трудоёмкая операция меняет состояние программы таким образом, что до следующей трудоёмкой операции обязательно пройдёт достаточно много мелких, тем самым «амортизируя» вклад трудоёмкой операции

1. *Назовите и опишите основные методы амортизационного анализа.*

А) Агрегирующий анализ

Вычисляется оценка сверху на время исполнения определённого количества операций, затем амортизационная сложность одной операции приравнивается к полученной оценке.

Б) Метод предоплаты

Каждой операции заранее приписывается амортизированная стоимость, которая может отличаться от её реальной стоимости. При этом более «дешёвые» операции обычно имеют амортизированную стоимость выше реальной, а более «дорогие» — ниже. За счёт этого при исполнении дешёвых операций накапливается некоторая сумма, которую можно «потратить» на то, чтобы выполнить операцию, чья амортизированная стоимость ниже реальной

В) Метод потенциалов

Накопленная сумма вычисляется как функция («потенциал») от состояния структуры данных. Амортизированная стоимость при этом равна сумме реальной стоимости и изменения потенциала

*3.Опишите структуру данных «Динамическая таблица (Dynamic Table)» и покажите, как выполняются операции добавления и удаления элементов.*

Динамическая таблица состоит из **последовательности строк.** Каждая запись может состоять из нескольких полей, одно из которых задаёт имя записи (ключ), остальные образуют тело записи

Память под элементы таблицы выделяется и освобождается по мере необходимости. Если начального размера таблицы недостаточно, создаётся новая структура большего размера с копированием всех прежних записей

***Добавление элемента в динамическую таблицу происходит следующим образом:***

1. Если есть свободное место, новый элемент записывается в свободную ячейку.
2. Если все элементы заняты, выделяется память под новую таблицу большего размера, в которую копируются записи из старой таблицы.
3. После добавления нового элемента старый размер таблицы уменьшается, а новый — увеличивается.

***Удаление элемента из динамической таблицы также имеет особенности, связанные с динамическим распределением памяти:***

1. Если элемент — последнее значение в таблице, его удаление сводится к уменьшению размера структуры на одну ячейку.
2. Если нужно удалить не последнее значение, все элементы правее удаляемого сдвигаются влево на одну позицию, а сама удаляемая запись удаляется

*4. Приведите анализ операций добавления и удаления элементов для структуры данных «Динамическая таблица (Dynamic Table)».*

***Анализ операции добавления элемента в динамическую таблицу****:*

Если в таблице есть свободное место, то новый элемент записывается в одну ячейку. Вычислительная сложность операции — O(1)

Если места нет, то может потребоваться расширение таблицы. В этом случае все элементы копируются в таблицу большего размера, после чего операция записывает новый элемент в расширенную таблицу. Вычислительная сложность расширения — O(n), где n — размер старой таблицы

Общая сложность n операций добавления — O(n²), если таблица расширяется каждый раз при отсутствии свободного места.

***Анализ операции удаления элемента из динамической таблицы****:*

Если элемент удаляется из пустой таблицы, то одна ячейка освобождается. Вычислительная сложность — O(1)

Если элемент удаляется из заполненной таблицы, то может потребоваться сдвинуть все элементы правее удаляемого влево на одну позицию. Сложность — O(n)

Общая сложность n операций удаления — O(n), если таблица не расширяется при удалении элементов. 

*5. Дайте определение абстрактного типа данных «Очередь с приоритетами (Priority Queue)».*

«Очередь с приоритетами» (priority queue) — это структура данных, представляющая собой обычную очередь, элементы которой упорядочены по их приоритету. От обычной очереди такая структура отличается тем, что при извлечении элемента из неё выбирается не тот, что был добавлен в очередь раньше всех, а элемент с максимальным приоритетом.

*6. На основе каких структур данных может быть реализован абстрактный тип данных «Очередь с приоритетами (Priority Queue)»?*

Он может быть реализован на :

**Неупорядоченный или упорядоченный массив**. Простейшая реализация, подходит для небольших очередей

**Связный список**. Используется в некоторых реализациях, но может быть неэффективным из-за необходимости обходить список при вставке элемента

**Куча**. Более эффективная структура данных, где операции вставки и извлечения выполняются за логарифмическое время. К кучам относятся, например, двоичная, биномиальная, фибоначчиева кучи.

*7. Покажите, как устроена структура данных «Двоичная пирамида (Binary Heap)». Опишите порядок выполнения операций SiftUp и SiftDown. Приведите оценку времени их работы*

**Двоичная куча (пирамида, сортирующее дерево, англ. binary heap)** — реализация очереди с приоритетами, использующая корневое дерево. Для неё выполнены три условия:

1. Значение в любой вершине не больше, чем значения её потомков.
2. У любой вершины не более двух сыновей.
3. Слои заполняются последовательно сверху вниз и слева направо, без «дырок».

**Операция siftUp** восстанавливает свойства кучи, если значение изменённого элемента уменьшается. Порядок выполнения:

1. Если элемент больше своего отца, то условие 1 соблюдено для всего дерева, и больше ничего делать не нужно.
2. Иначе элемент меняется местами с отцом.
3. После этого выполняется siftUp для этого отца. Иными словами, слишком маленький элемент всплывает наверх.

**Операция siftDown** используется для восстановления свойства кучи с конца: постепенно увеличивается суффикс, на котором выполняется это свойство, с помощью просеивания вниз.

**Оценка времени работы операции siftUp**: процедура выполняется за время O(log n)

**Оценка времени работы операции siftDown**: в худшем случае новый корень нужно поменять местами с его ребёнком на каждом уровне, пока он не достигнет нижнего уровня кучи. Это означает, что операция удаления имеет сложность времени, связанную с высотой дерева, или O(log n)

*8. Опишите, как на основе массива построить двоичную пирамиду за время O(n).*

Для построения пирамиды из произвольного массива используется метод "просеивания вниз", применяемый к элементам, начиная с последнего нелистового узла до корня.

**Алгоритм**

Начальная точка: начинаем с последнего нелистового узла

Просеивание вниз: для каждого узла выполняем операцию sift down

Последовательность обработки: узлы обрабатываются в обратном порядке, от последнего нелистового до корня

**Ключевые моменты**

Листовые узлы (около половины элементов) уже удовлетворяют свойствам пирамиды и не требуют обработки

Высота обработки: каждый узел на уровне h требует не более h операций сравнения/обмена

Суммарная сложность: благодаря тому, что большинство узлов находятся на нижних уровнях, общее время составляет O(n)

*9. Приведите псевдокод операций Enqueue, Dequeue и Increase/DecreaseKey*

*для очереди с приоритетами, реализованной на основе двоичной пирамиды.*

**Псевдокод операции Enqueue**:

1. Вставить элемент с приоритетом в конец двоичной пирамиды
2. Выполнить функцию heapify() для восстановления свойства кучи

**Псевдокод операции Dequeue**:

1. Извлечь элемент с максимальным приоритетом из двоичной пирамиды
2. Заменить значение элемента на значение последнего узла в дереве
3. Удалить последний узел
4. Выполнить функцию heapify() над новой вершиной.

**Псевдокод операции IncreaseKey**:

1. Увеличить приоритет элемента, для чего переместить его вверх по пирамиде, пока он не окажется на вершине
2. После увеличения выполнить функцию heapify() от вершины, чтобы восстановить свойство кучи

**Псевдокод операции DecreaseKey**:

1. Уменьшить приоритет элемента, для чего переместить его вниз по пирамиде, пока он не окажется на листе
2. После уменьшения выполнить функцию heapify() от корня, чтобы восстановить свойство кучи