Базовые структуры данных

**Структура данных —** это контейнер, который хранит данные в определенном макете. Этот «макет» позволяет структуре данных быть эффективной в некоторых операциях и неэффективной в других.

**Структуры данных делятся на 2 типа:**

**Линейные,** элементы образуют последовательность или линейный список, обход узлов линеен. Примеры: Массивы. Связанный список, стеки и очереди.

**Нелинейные,** если обход узлов нелинейный, а данные не последовательны. Пример: граф и деревья.

**Основные структуры данных:**

1. Массивы
2. Стеки
3. Очереди
4. Связанные списки
5. Графы
6. Деревья
7. Хэш таблицы

Массивы

**Массив** — это самая простая и широко используемая структура данных. Другие структуры данных, такие как стеки и очереди, являются производными от массивов. Каждому элементу данных присваивается положительное числовое значение (индекс), который соответствует позиции элемента в массиве. Большинство языков определяют начальный индекс массива как 0.

**Массивы делятся на:**

**Одномерные** – каждый элемент лежит друг за другом в одну строку.

**Многомерный** – это массивы внутри массивов.

**Основные операции над массивами:**

* **Insert** - вставляет элемент по заданному индексу
* **Get** - возвращает элемент по заданному индексу
* **Delete** - удаление элемента по заданному индексу
* **Size** - получить общее количество элементов в массиве

Стеки

**Стек** — абстрактный тип данных, представляющий собой список элементов, организованных по принципу LIFO (англ. last in — first out, «последним пришёл — первым вышел»). Это не массивы. Это очередь.

Примером стека может быть куча книг, расположенных в вертикальном порядке. Для того, чтобы получить книгу, которая где-то посередине, вам нужно будет удалить все книги, размещенные на ней. Так работает метод LIFO (Last In First Out). Функция «Отменить» в приложениях работает по LIFO.

**Основные операции над стеками:**

* **Push** - вставляет элемент сверху
* **Pop** - возвращает верхний элемент после удаления из стека
* **isEmpty** - возвращает true, если стек пуст
* **Top** - возвращает верхний элемент без удаления из стека

Очереди

**Очередь** — подобна стеку, хранит элемент последовательным образом. Существенное отличие от стека – использование FIFO (First in First Out) вместо LIFO.

Пример очереди – очередь людей. Последний занял последним и будешь, а первый первым ее и покинет.

**Основные операции над очередями:**

* Enqueue - вставляет элемент в конец очереди
* Dequeue - удаляет элемент из начала очереди
* isEmpty - возвращает значение true, если очередь пуста
* Top - возвращает первый элемент очереди

Связанный список

**Связанный список** – массив, где каждый элемент является отдельным объектом и состоит из двух элементов – данных и ссылки на следующий узел.

Принципиальным преимуществом перед массивом является структурная гибкость: порядок элементов связного списка может не совпадать с порядком расположения элементов данных в памяти компьютера, а порядок обхода списка всегда явно задаётся его внутренними связями.

**Связные списки делятся на три вида:**

**Однонаправленный** - каждый узел хранит адрес или ссылку на следующий узел в списке и последний узел имеет следующий адрес или ссылку как NULL.

**Двунаправленный** - две ссылки, связанные с каждым узлом, одним из опорных пунктов на следующий узел и один к предыдущему узлу.

**Круговой** - все узлы соединяются, образуя круг. В конце нет NULL. Циклический связанный список может быть одно-или двукратным циклическим связанным списком.

**Основные операции над связными списками:**

* **InsertAtEnd** - вставка заданного элемента в конец списка
* **InsertAtHead** - вставка элемента в начало списка
* **Delete** - удаляет заданный элемент из списка
* **DeleteAtHead** - удаляет первый элемент списка
* **Search** - возвращает заданный элемент из списка
* **isEmpty** - возвращает True, если связанный список пуст

Графы

**Граф** — это набор узлов (вершин), которые соединены друг с другом в виде сети ребрами (дугами).

**Графы делятся на два вида**:

**Ориентированный** - ребра являются направленными, т.е. существует только одно доступное направление между двумя связными вершинами.

**Неориентированные** - к каждому из ребер можно осуществлять переход в обоих направлениях.

**Встречаются в таких формах как:**

* Матрица смежности
* Список смежности

**Общие алгоритмы обхода графа:**

* Поиск в ширину – обход по уровням
* Поиск в глубину – обход по вершинам

Деревья

**Дерево** — это иерархическая структура данных, состоящая из узлов (вершин) и ребер (дуг). Деревья, по сути, связанные графы без циклов.

**Типы деревьев**

* N дерево
* Сбалансированное дерево
* Бинарное дерево
* Дерево Бинарного Поиска
* AVL дерево
* 2-3-4 деревья

Три способа обхода дерева

* В прямом порядке (сверху вниз) — префиксная форма.
* В симметричном порядке (слева направо) — инфиксная форма.
* В обратном порядке (снизу вверх) — постфиксная форма.

Хэш таблицы

**Хеширование** — это процесс, используемый для уникальной идентификации объектов и хранения каждого объекта в заранее рассчитанном уникальном индексе (ключе).  
Объект хранится в виде пары «ключ-значение», а коллекция таких элементов называется «словарем». Каждый объект можно найти с помощью этого ключа.  
По сути это массив, в котором ключ представлен в виде хеш-функции.

Эффективность хеширования зависит от:

* Функции хеширования
* Размера хэш-таблицы
* Метода борьбы с коллизиями