Данные

ОСНОВЫ ОСНОВ

Что такое данные?

Данные — это информация в формализованном виде, пригодном для передачи, интерпретации и обработки.

В современных компьютерных системах, построенных на архитектуре фон Неймана, и данные, и инструкции для их обработки могут находиться в одной и той же области памяти, что означает, что программы и функции — это тоже данные.

Что за архитектура такая?

Джон фон Нейман — математик и физик, который, помимо прочих своих заслуг и достоинств, принимал участие в создании первого в мире лампового компьютера ЭНИАК.

Группа учёных, работавшая над созданием ЭНИАКа в 40-х годах XX века, сформулировала как минимум два основополагающих для всей современной айтишечки принципа, но по нелепой случайности на обложке отчёта было только имя фон Неймана, что и привело к появлению соответствующего названия.

Принцип 1

Все числа нужно представлять в двоичном виде. Такой подход значительно упростил выполнение сложных арифметических и логических операций, что привело к тому, что впоследствии вообще все данные стали представлять в виде набора двоичных чисел, не разбираясь, текст это, числа, изображения или что угодно ещё.

Принцип 2

Раз уж у нас всё двоичные числа, то инструкции для обработки этих чисел тоже можно представить такими числами. А если нет разницы, то можно инструкции хранить в той же памяти, а не отдельно, что открывает гигантские возможности, которые нами уже воспринимаются как нечто само собой разумеющееся.

Например, с этого момента команды программы могут быть результатом работы другой программы, и именно этому явлению мы обязаны удовольствием кодить на питоне, например.

И что?..

И то: поскольку данные могут быть чем угодно — числами, последовательностями байтов, указателями на области памяти, исполняемыми инструкциями, встаёт вопрос о том, как их использовать, чтобы страдать поменьше, а эффективность была повыше. Решение этого вопроса породило в Computer Science такое понятие, как структуры данных.

«Плохие программисты думают о коде. Хорошие программисты думают о структурах данных и их взаимосвязях»

Линус Торвальдс

Структуры данных — это фундаментальная концепция в Computer Science, описывающая, как можно организовывать и обрабатывать данные. Значительная часть структур данных перекочевала в компьютерные науки из других областей, как правило — математики.

Структур данных немало, но мы разберём только наиболее употребительные. Их можно разбить на две группы: примитивные и составные.

Вроде ж были типы, а не структуры?

Тип данных — это реализация концепции структуры данных, которая может несколько отличаться в разных языках программирования.

Тем не менее, тип данных обладает общей для всех языков особенностью и характеризует, какие допустимые значения могут принимать данные этого типа, и какие операции можно с ними выполнять.

Примитивные структуры данных

К этой группе относятся одни из наиболее очевидных не-математикам структур:

- целые числа (integers);
- числа с плавающей запятой (floating-point numbers);
- логические значения (booleans);
- символы (characters).

Из этих структур отдельно стоит остановиться на символах — они не представлены в питоне самостоятельным типом данных, вместо этого используются строки единичной длины.

Составные структуры данных

Составные структуры делятся на несколько подгрупп:

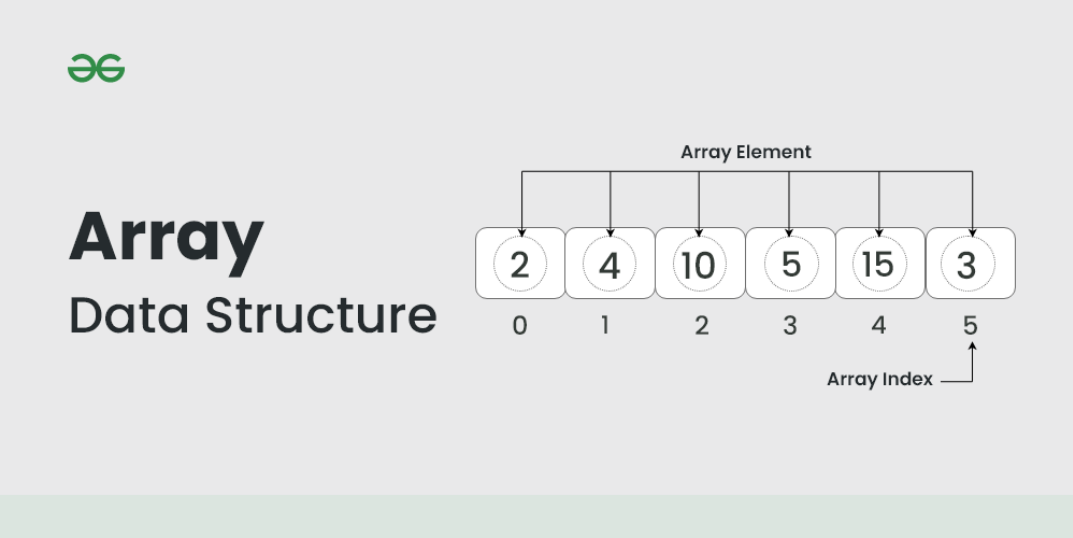
- последовательности (sequences);
- деревья (trees);
- графы (graphs);
- хеш-таблицы (hash tables);
- очереди (queues);
- множества (sets);
- и многое другое.

Последовательности

Последовательности — это упорядоченные коллекции элементов, в которых каждому элементу присваивается позиция или индекс.

Бывают нескольких видов:

- массивы (arrays);
- кортежи (tuples);
- связанные списки (linked lists);
- строки (strings).



Array

Массивы — это коллекции элементов одного типа, хранящиеся в непрерывном блоке памяти, за счёт чего скорость доступа по индексу просто бешеная. Во многих языках программирования массивы создаются фиксированной длины, но бывают и динамические реализации.

Массивы в Python

В питоне массивы как раз динамические, они представлены в двух вариантах:

- **list()** — изменяемая коллекция элементов разных типов;
- **array.array()** — изменяемая коллекция элементов одного типа.

Проблема динамических массивов в том, что при создании массива выделяется больший блок памяти чем нужен, и когда он заканчивается, выделяется новый блок и массив копируется в него.

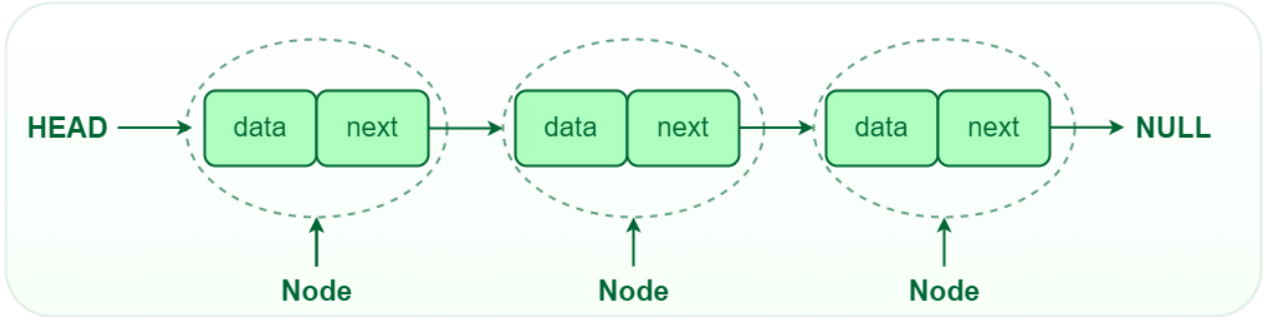
Это порождает неочевидные проблемы с расходованием памяти, потому что старый массив никуда не девается, пока не пройдёт сборщик мусора, плюс тратится время на копирование.

Tuple

Кортежи — это неизменяемые коллекции элементов. Основной поинт именно в неизменяемости, каноничный пример — описание координат: (x, y).

Реализация в питоне — tuple().

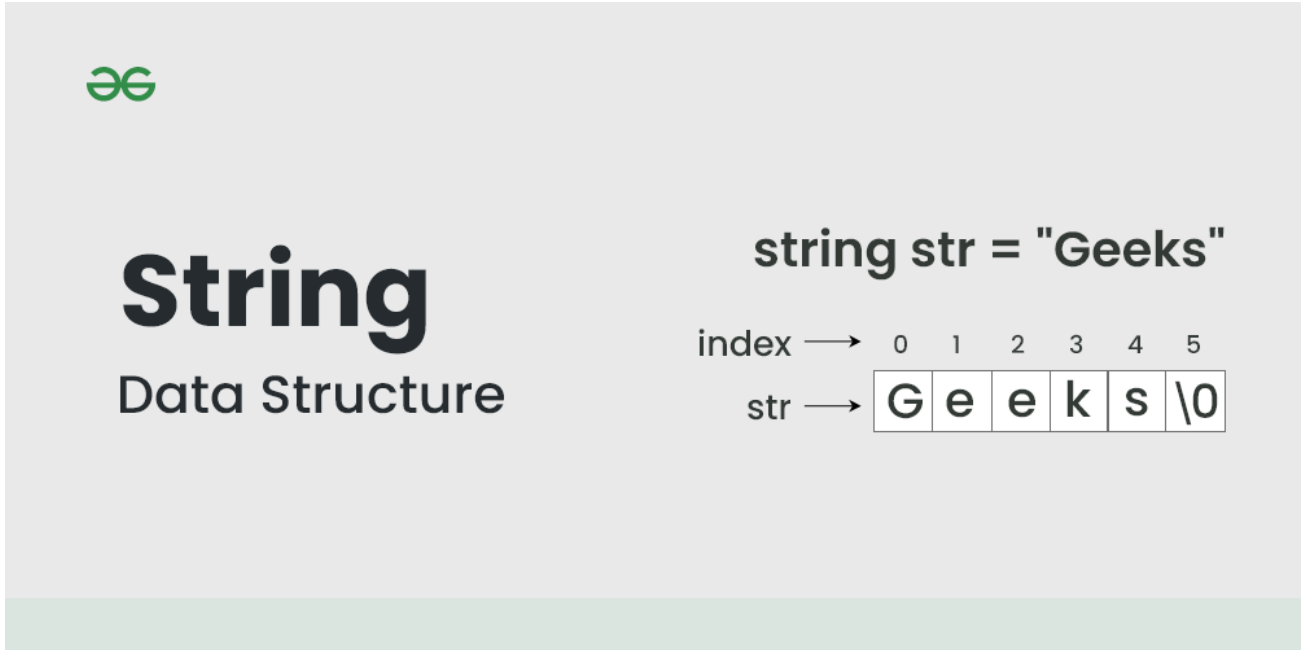
Linked list



Связанные списки — это структуры данных, в которых элементы указывают друг на друга.

Бывают односвязными и двусвязными: в односвязных каждый элемент указывает на следующий, в двусвязных — и на следующий, и на предыдущий.

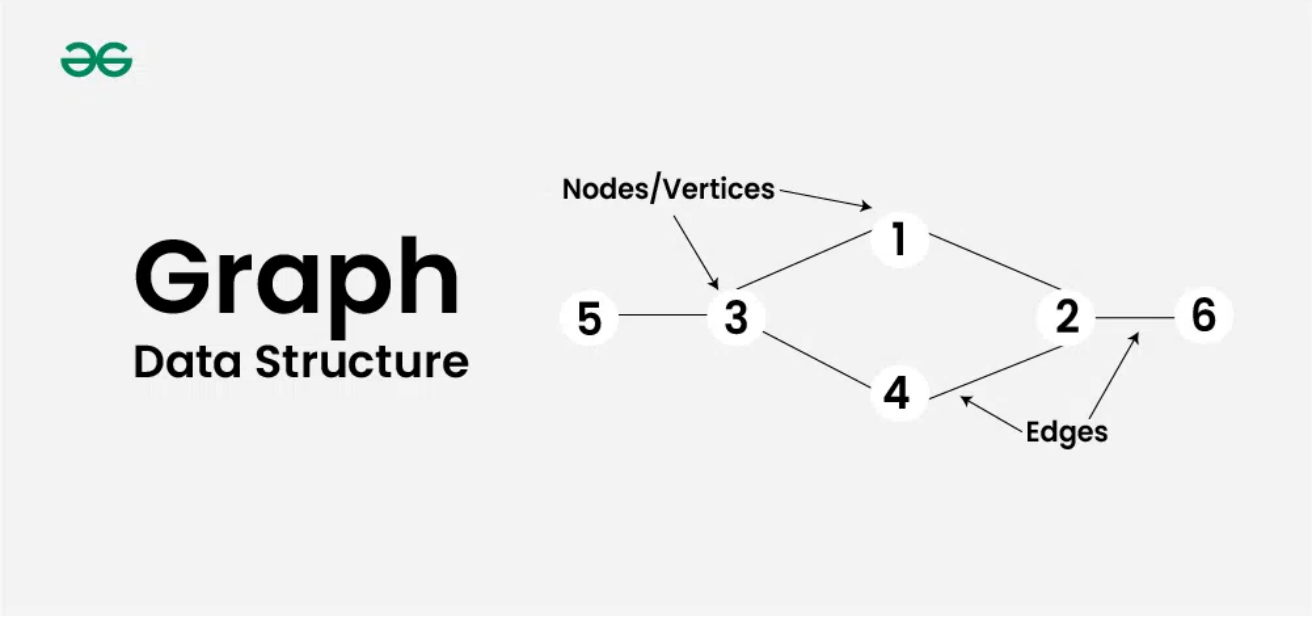
В питоне не представлены, но можно написать свою реализацию.



String

Строки — это неизменяемые последовательности символов. Несмотря на то, что они обладают некоторыми свойствами массивов, это отдельная структура.

В общем, больше ничего особо и не скажешь.



Graph

Графы — это структура данных, представляющая собственно граф — математическую абстракцию, состоящую из вершин и соединяющих их рёбер.

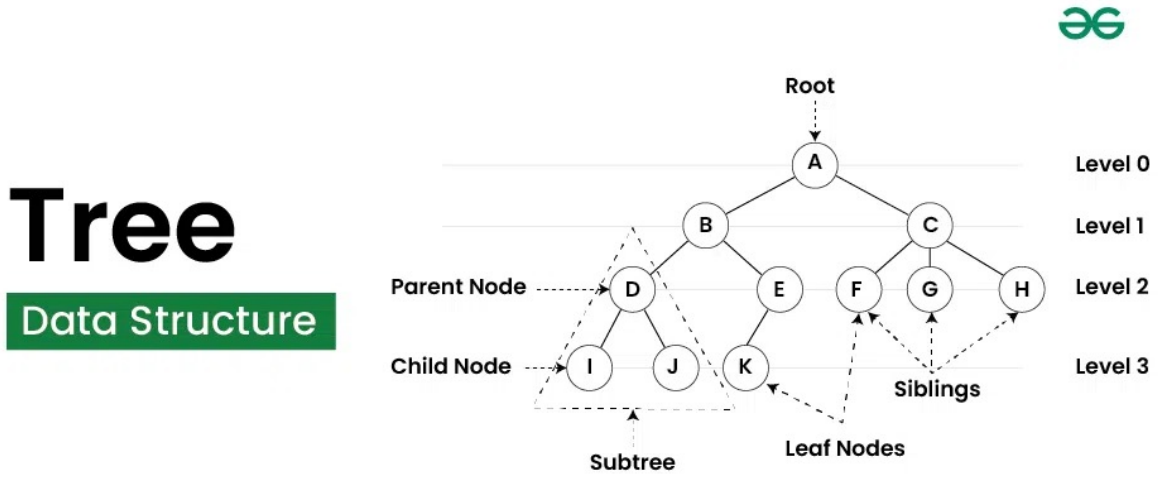
Весьма подробно описанные теорией графов, графы представляют великолепный инструмент для изучения сложных взаимосвязей и применяются для моделирования сетей, маршрутизации, расчёта социальной динамики и многого другого.

Типы графов

- **Направленные** — рёбра графа имеют направление и связь между вершинами возможна только в этом направлении.
- **Ненаправленные** — связь между вершинами возможна в обоих направлениях.
- **Взвешенные** — рёбра имеют веса, что определяет порядок обхода графа.
- **Невзвешенные** — все рёбра равны.
- **Циклические** — содержат как минимум один путь, начинающийся и заканчивающийся в одной точке.

В питоне самостоятельным типом не представлены, но могут быть реализованы вручную.

Tree

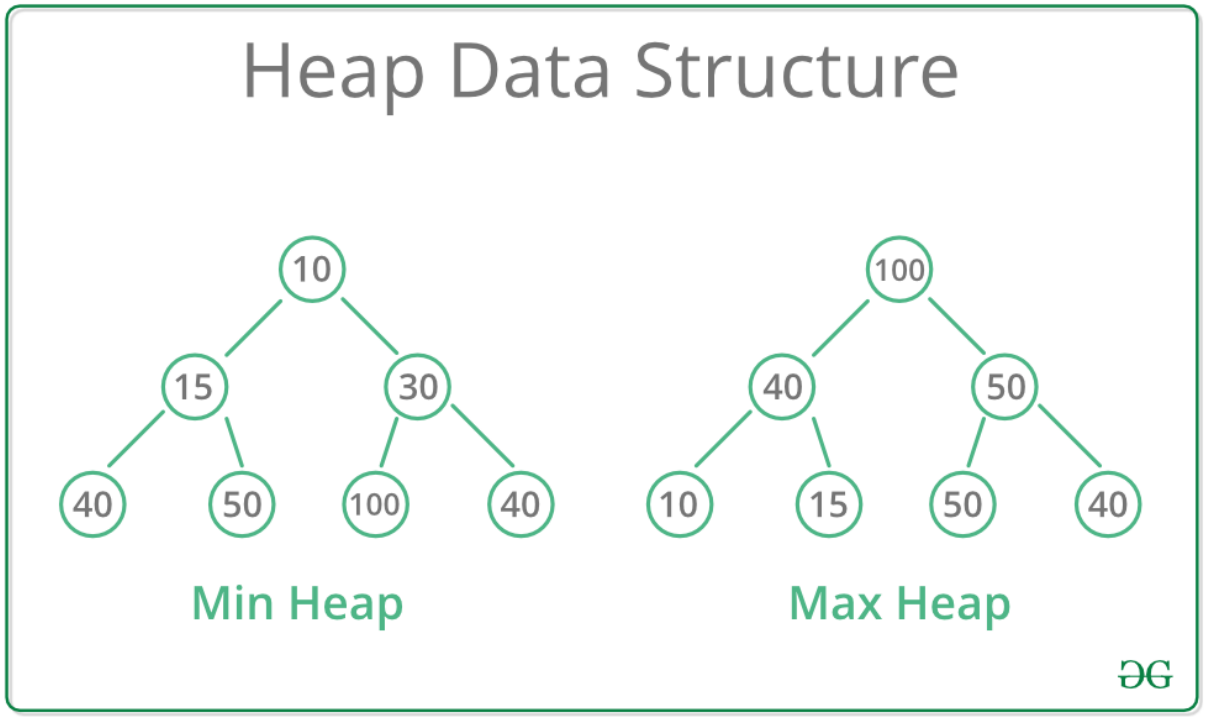


Деревья — частный случай графов, но они имеют чёткую иерархию и всегда начинаются от корня.

Основные части дерева:

- **Корень (root)**: Самый верхний узел дерева, не имеющий родителя.
- **Узел (node)**: Элемент дерева, содержащий данные. Узлы могут иметь потомков.
- **Лист (leaf)**: Узел, не имеющий потомков.
- **Ребро (edge)**: Связь между узлами.
- **Поддерево (subtree)**: Дерево, являющееся частью другого дерева.
- **Высота (height)**: Максимальное количество ребер от корня до листа.

В питоне не представлены, могут быть реализованы руками.



Heap tree

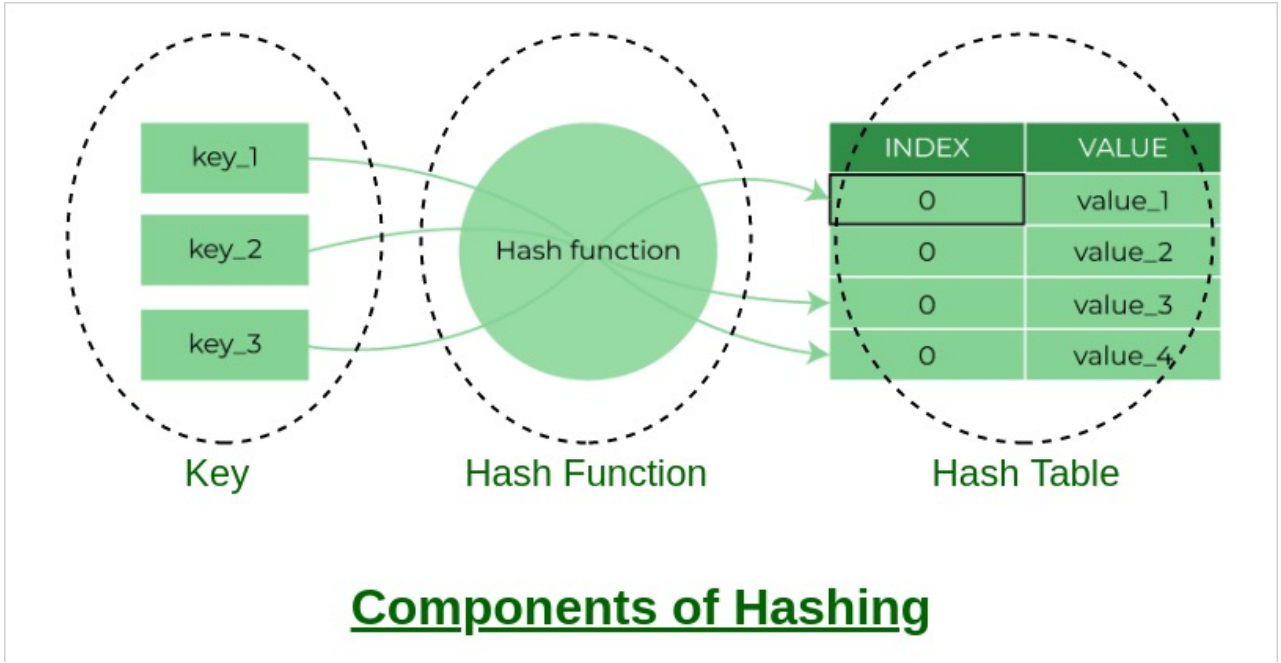
Отдельный вид дерева — куча (heap).

Куча — это полное бинарное (не более двух потомков у каждого узла) дерево, в котором все уровни, кроме последнего, заполнены полностью, а узлы последнего уровня заполняются слева направо.

- **Миникуча (min-heap):** В любой момент родительский узел меньше или равен своим дочерним узлам.

- **Максикуча (max-heap):** В любой момент родительский узел больше или равен своим дочерним узлам.

В питоне куча представлена модулем heapq, который реализует только миникучу.



Hash table

Хеш-таблицы позволяют хранить и быстро извлекать данные по ключу. Они используют хеш-функции для преобразования ключей в индексы, которые указывают на позиции в массиве, где хранятся значения.

Основные компоненты хеш-таблицы:

- **Ключи (keys)**: уникальные идентификаторы, по которым осуществляется доступ к значениям.
- **Значения (values)**: данные, ассоциированные с ключами.
- **Хеш-функция (hash function)**: функция, которая преобразует ключ в индекс массива.
- **Массив (array)**: структура, в которой хранятся значения.

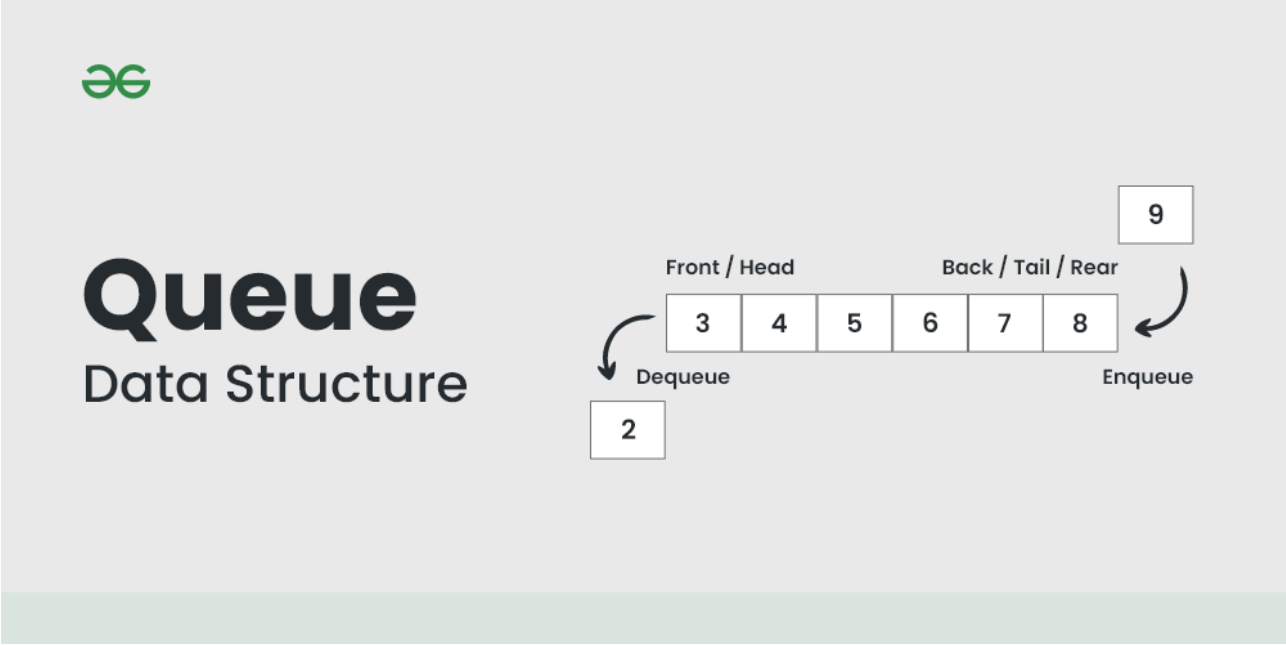
Принцип работы

1. **Хеширование**: ключи преобразуются в индексы массива с помощью хеш-функции.

2. **Разрешение коллизий**: коллизии возникают, когда два ключа хешируются в один и тот же индекс. Существует несколько методов разрешения коллизий, таких как цепочки (chaining) и открытая адресация (open addressing), но пока туда можно не лезть.

В питоне хеш-таблицы представлены вот какими типами:

- **dict()** — обычная реализация хеш-таблицы;
- **set()** — хеш-таблица для хранения уникальных элементов.

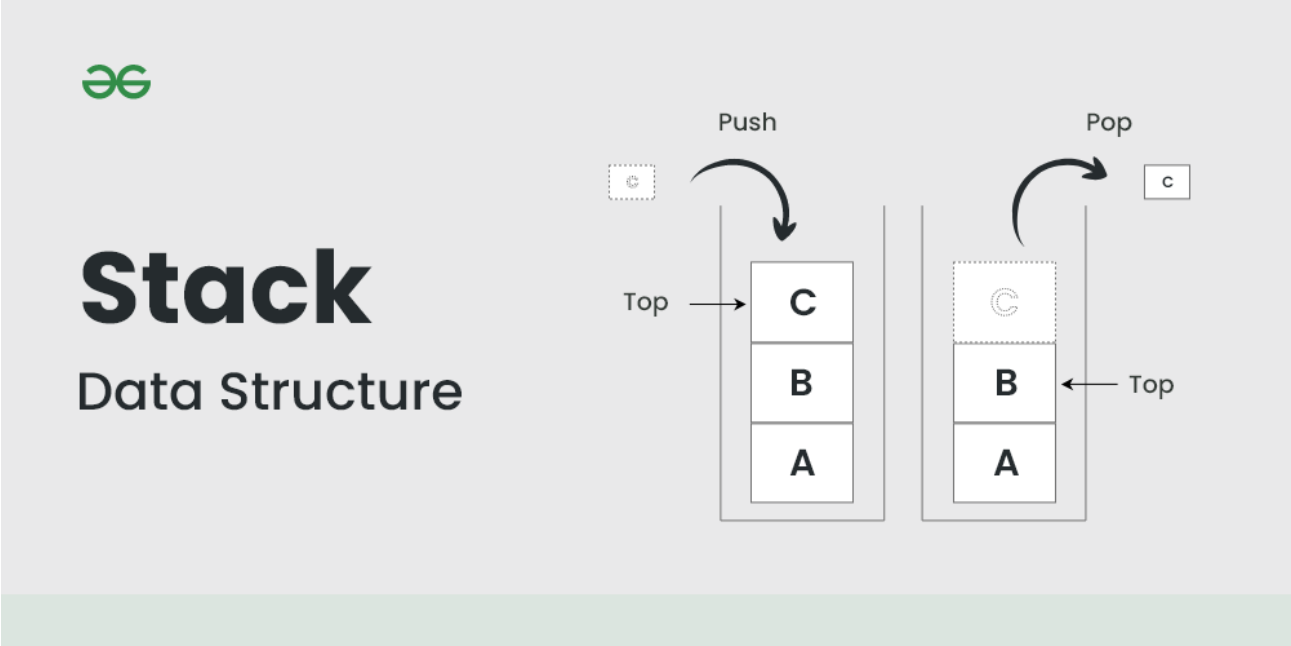


Queue

Очереди — это, собственно, очереди и есть. Бывают разных типов: очередь, стек, двунаправленная очередь, и пр.

Очередь (`queue`) — обычная очередь, как в магазине.

Реализует порядок FIFO (First In, First Out) — кто первым пришёл, того первым и вынули. Представлена классом `queue.Queue`, но можно и свою написать.



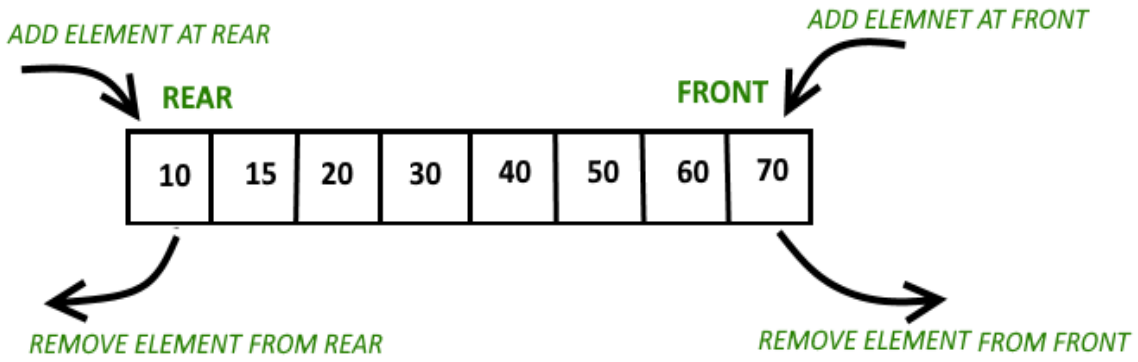
Stack

Вид очереди, который реализует порядок LIFO (Last In, First Out).

Обычно используется для обхода графов или реализации любого другого перемещения «вперёд-назад».

В питоне реализуется только руками.

Dequeue



Двусторонняя очередь позволяет добавлять и удалять элементы с любого конца.

Реализация есть в модуле collections.

Set

Множества — это неупорядоченные структуры уникальных элементов. Часто реализуются через хеш-таблицы, но могут быть и самобалансирующимися деревьями, и списками, и битовыми массивами, и чем только не.

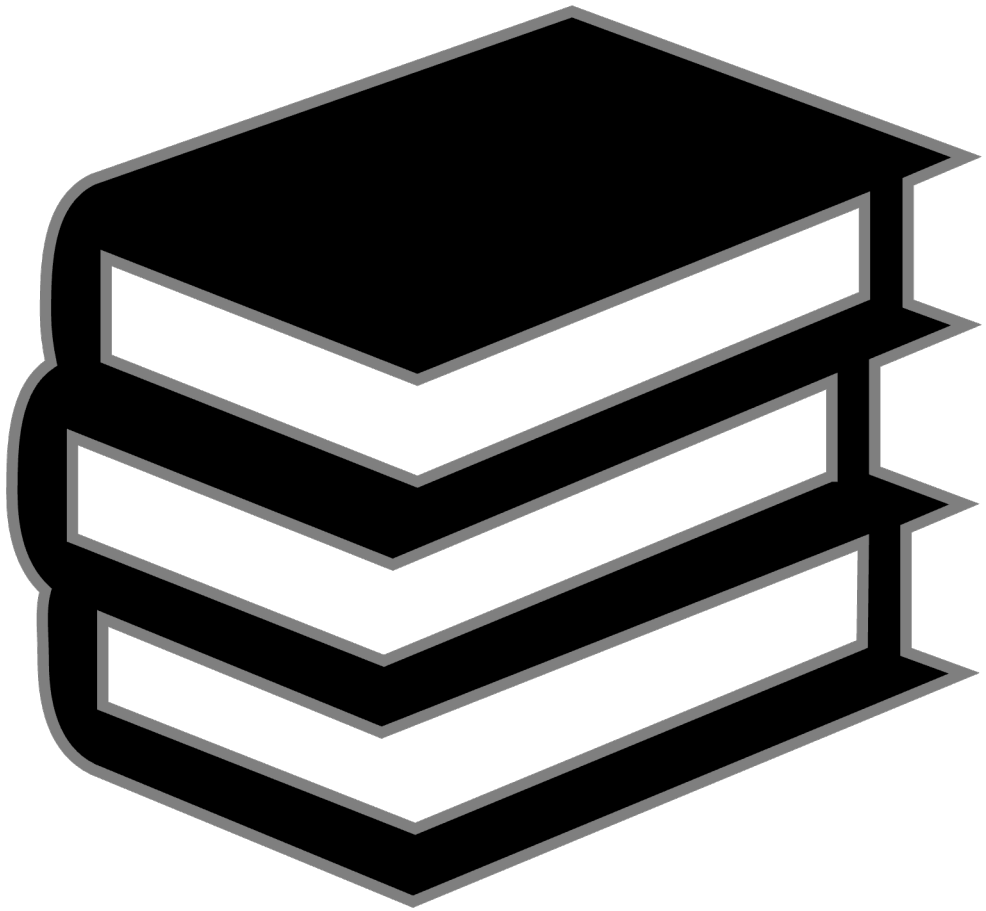
Отличие от любых других коллекций — уникальность элементов и поддержка некоторых свойств, присущих математическим множествам, описанным в теории множеств: пересечения, объединения, исключения, и так далее.

В питоне множество — это `set()`, реализованный на хеш-таблице.

Заключение

Важно понимать, что составных структур данных гораздо больше почти в каждом из рассмотренных типов. Существуют сложносоставные структуры, структуры для поиска, структуры для представления 3D-моделей и прочее, прочее, прочее.

Все их знать редко кому нужно, но понимать, что всё — данные, а значит, их можно организовывать в структуры, и эти структуры с высокой долей вероятности уже изобретены — необходимо.



Источники

- Википедия для справочного материала.
- GeeksForGeeks для картинок.