AVL Tree: Approximately balanced binary search tree

- For any AVL Tree the height of its left sub tree and right sub tree do not vary by more than one | h(LST) - h(RST)| <= 1 (The height is the number of levels from a chosen node on either the left or right side)
- The node from which a new number is added is called the node of imbalance
- 4 Cases or Imbalance (Determined by where exactly the insertion is, and which node becomes “deeper” than the rest)

Memory Hierarchy: DB systems aim to minimize HDD/SDD accesses

- SDD/HDD use lots of storage and are slow
- Hard drive stores in “blocks”:
    - If we have a 2048 byte block and we want to access a 64 bit it (8 bytes), we stlll must search the whole 2048 byte block
- Consider a sorted array of 128 ints (1024 bytes)
    - If its all in one block, binary search is much faster than even searching for 1 out of 2 ints where the ints are in 2 different blocks (less disk accesses)
    - Remember we don’t get too much control, but the general idea is to reduce the number of different disk accesses

B+ Tree

    - Minimizes the height of the tree
- Node structure of B+ Tree:
    - Maximum of 2 children per node
    - All nodes except the root must be at least half full
    - Leaves are stored as DLL
- Internal nodes only store keys and pointers to other values
- Leaf nodes can access values too
    - The more things we put into nodes, the less space there is for keys, resulting in a deeper tree, which we don’t want
    - IIn B trees, nodes store keys and values, but in B+ trees, only leaf nodes can access values
- Splitting a B+ tree
    - Tree only gets deeper once the root node needs to split
    - Smallest value of a new leaf node gets copied to the parent node

        - New node always contains the greater of the values we are splitting
    - When you split an internal node, greatest value og the new node gets moved up, not copied (only copied when leaf splits)

NoSQL and Alternative DBs

- NoSQL means Not Only SQL, refers to non-relational databases that store data in a non-tabular format
- CAP Theorem
    - Consistency: Every user of the database has an identical view of the data at any given instant
    - Availability: In the event of a failure, the database system remains operational
    - Partition Tolerance: Database continues to function even when there is a communication break between nodes
- If htere is a break in communication between two nodes, one can either
    - Allow nodes to get out of sync (giving up consistency), or
    - Consider the system to go “down” (giving up availability)
- All options:
    - CA: Data is consistent and available, but if there is a partition, data will be out of sync
    - CP: Data is consistent, and partition tolerance is maintained by making the system unavailable when a node goes down
    - AP: Nodes remain online even if they can’t communicate with each other, but no guarantee that the nodes will all have the same data at the same times

RDBMS like MySQL use CA

NoSQL like Redis and MongoDB use CP

Key Value Stores

- Comparatively simpler than tabular RDBMS
- Lends itself to easy CRUD operations and API creation
- KV stores are usually deployed as in-memory Databases

- Getting a value is typically O(1) because hash tables or a similar structure is under the hood