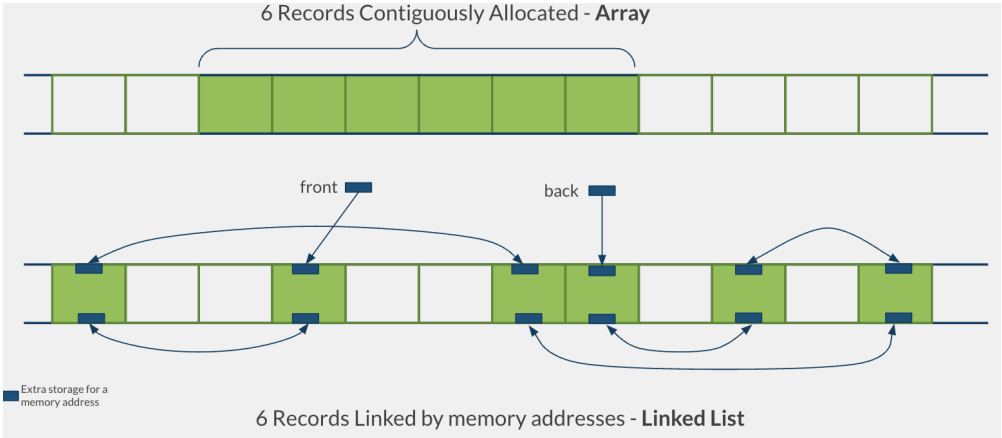
Lecture 2 - Foundations

Searching

- Baseline for efficiency: **Linear Search**
    - Start at beginning of list, progresses from first to last element until end or a targeted element found
- Record - row of a table
- Collection - set of records of same entity type; table
- Search key - value for an attribute from entity type

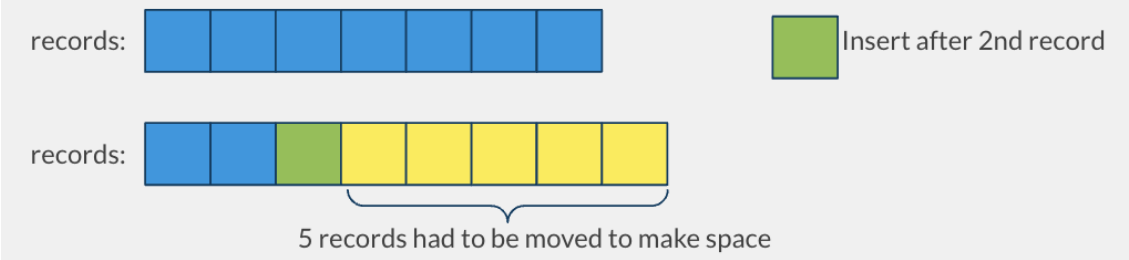
Lists of Records

- Need ***n*x*** bytes of memory if each record takes up ***x*** bytes of memory for ***n*** records
- **Contiguously Allocated List** - all ***n*x*** bytes are allocated as a single chunk of memory
- **Linked List**
    - Each record needs ***x*** bytes + space for links
    - Memory addresses links individual records in a chain

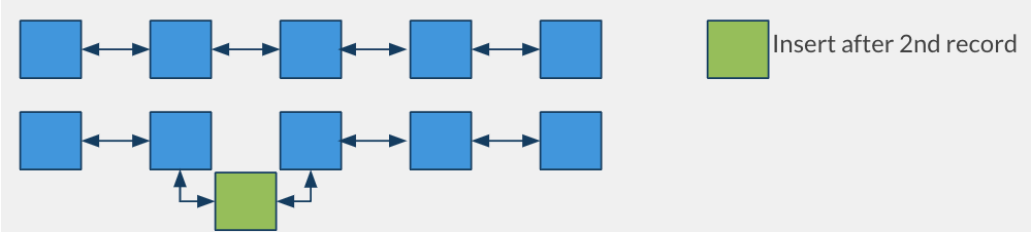


Pros and Cons

- Arrays are faster for random access but slow for inserting anywhere but the end

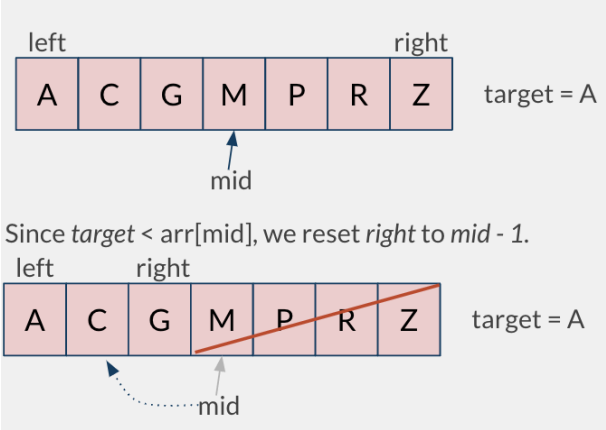


- Linked lists are faster for inserting anywhere in list but slower for random access



Binary Search

- Input - array of values in sorted order
- Output - location (index) of where target is located
- Can't perform binary search on unsorted array



Time Complexity

- Linear Search
  - Best case - target found at first element; only 1 comparison
  - Worst case - target not in array; n comparisons
- Binary search
  - Best case - target found at mid; 1 comparison inside the loop
  - Worst case - target not in the array; log base2 n comparisons

Database Searching

- Want to search for a specific specialVal?
  - Only option is linear scan of that column
- Can't store data on disk sorted by both id and specialVal (at the same time) b/c data would have to be duplicated
- Need an external data structure to support faster searching by specialVal than a linear scan

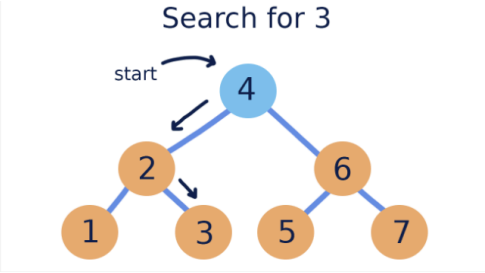
Options - database indexes

1. Array of tuples (specialVal, rowNumber) sorted by specialVal
   a. Use Binary Search
   b. But every insert into a sorted array is slow

2. A linked list of tuples(specialVal, rowNumber) sorted by specialVal
    a. Searching is slow, linear scan required

Binary Search Tree

- Fast insert fast search
- A binary tree where every node in the left subtree is less than its parent and every node in the right subtree is greater than its parent



- **Tree Traversals**
    - Pre Order
    - Post Order
    - In Order
    - **Level Order**
        - Know output of level order traversal of a binary search tree: 23, 17, 43, 20, 31, 50
        - How to process the tree algorithmically with the same output^: start at the root, next should be what is left of the root, temp store next 2 elements to process

HW: deepest level, reverse traversal output

Lecture 3 - AVL Tree

After inserting a node:

- Work your way back up the tree from the position where you just added a node. (This could be quite simple if the insertion was done recursively.) Compare the heights of the left and right subtrees of each node. When they differ by more than 1, choose a rotation that will fix the imbalance.
    - Note that comparing the heights of the left and right subtrees would be quite expensive if you didn't already know what they were. The solution to this problem is for each node to store its height (i.e., the height of the subtree rooted there). This can be cheaply updated after every insertion or removal as you unwind the recursion.
- The rotation is chosen considering the two links along the path *below* the node where the imbalance is, heading back down toward where you inserted a node. (If you were wondering where the names LL, RR, LR, and RL come from, this is the answer to that mystery.)

    - If the two links are both to the left, perform an LL rotation rooted where the imbalance is.
    - If the two links are both to the right, perform an RR rotation rooted where the imbalance is.
    - If the first link is to the left and the second is to the right, perform an LR rotation rooted where the imbalance is.
    - If the first link is to the right and the second is to the left, perform an RL rotation rooted where the imbalance is.
- **Single Rotations (LL or RR)**: In these cases, the rotation directly involves a node and its child. For example:
    - LL (Left-Left) rotation affects a node and its left child.
    - RR (Right-Right) rotation affects a node and its right child.
- **Double Rotations (LR or RL)**: These rotations involve more levels:
    - LR (Left-Right) rotation affects a node, its left child, and the left child's right child.
    - RL (Right-Left) rotation affects a node, its right child, and the right child's left child.

Practical A:

- Inverted index

Beyond Relational Model

NoSQL and KV DBs

Optimistic Concurrency

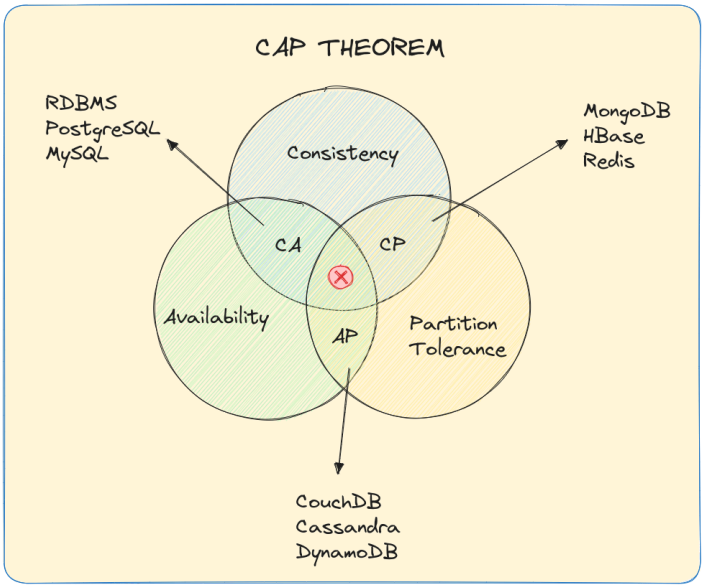
- Low conflict systems (backups, analytical dbs, etc.)
    - Read heavy systems
- High conflict systems - rolling back and re-executing transactions that encounter a conflict -> less efficient

NoSQL

- “Not only SQL”

CAP Theorem Review

- Can have 2 but not 3 of the following:
    - **Consistency** - every user of the DB has an identical view of the data at a given instant
    - **Availability** - in the event of a system failure, db system remains operational
    - **Partition Tolerance** - the db can maintain operations in the event of the network failing between 2 systems



ACID Alternative for Distribution Systems

- Alternative is BASE model
- **Basically Available** - guarantees availability of the data (per CAP), but response can be “failure”/”unreliable”
- **Soft State** - state of the system can change overtime even without input, changes could be a result of eventual consistency
- **Eventual Consistency** - system will eventually become consistent

Categories of NoSQL DBs

- Document databases
- Vector databases
- Key-value databases

Key Value Stores
Key = value

- Key value stores are designed around
    - Speed
        - Usually deployed as an in-memory DB
        - Retrieving a value given its key is typically a O(1) op
    - Scalability
        - Horizontal scaling is simple - add more nodes
        - Only guarantee is that all nodes will eventually converge on the same value

KV DS Use Cases

- EDA/Experimentation results store
- Feature store
- Model monitoring

Redis DB

- Remote Directory Server
- Open source in-memory db
- Sometimes called a data structure store
- Redis docker desktop

Document DBs & Mongo DB

- Document database - non relational database that stores data as structured documents, usually in JSON
- BSON (binary json) - binary encoded serialization of a JSON like document structure
    - Lightweight, traversable, efficient
- XML (eXtensible Markup Language) - precursor to JSON as data exchange format

PyMongo

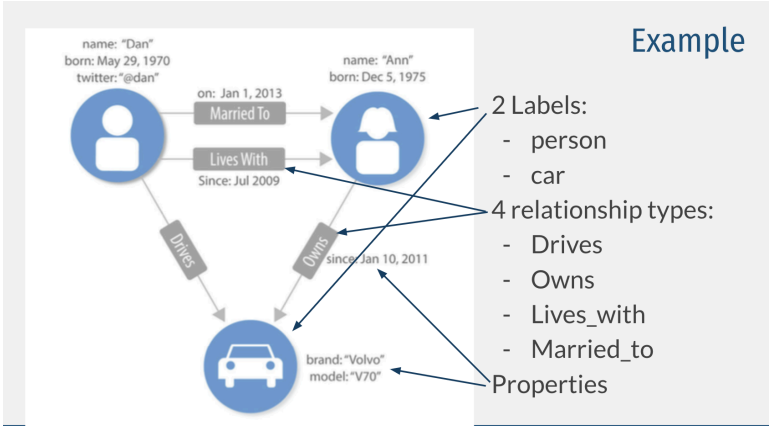
- A python library for interfacing with MongoDB instances

Introduction to the Graph Data Model

What is a graph database?

- Data model based on the graph data structure
- Composed of nodes and edges
    - Edges connect nodes
    - Each uniquely identified
    - Supports queries based on graph-oriented operations (traversals, shortest path)

What is a graph?



- Path - ordered sequence of nodes connected by edges in which no nodes or edges are repeated

Flavors of Graphs
- Connected vs Disconnected - there is a path between any two nodes in the graph
- Weighted vs Unweighted - edge has a weight property (important for some algos)
- Directed vs Undirected - relationships (edges) define a start and end node
- Acyclic vs Cyclic - contains no cycles

Types of Graph Algo - Pathfinding
- Finding the shortest path between two nodes if one exists is most common operation
- Shortest means fewest edges or lowest weight
- Average Shortest Path can be used to monitor efficiency and resiliency of networks
- Minimum spanning tree, cycle detection, max/min flow

Types of Graph Algo - Centrality and Community Detection
- Centrality - determining which nodes are “more important” in a network compared to other nodes
- Community Detection - evaluate clustering or partitioning of nodes of a graph and tendency to strengthen or break apart

Docker Compose & Neo4j

Neo4j
- Cypher language
- APOC Plugin - extends functionality of cypher
- Graph data science plugin

Docker Compose
- Supports multi-container management
- 1 command can be used to start, stop, or scale a number of services at once

.env Files
- .env files - stores a collection of environment variables
- Good way to keep environment variables for different platforms separate

Ollama pull mistral:latest
VS code terminal
Cd src
Python ingest.py
Separate terminal ollama list