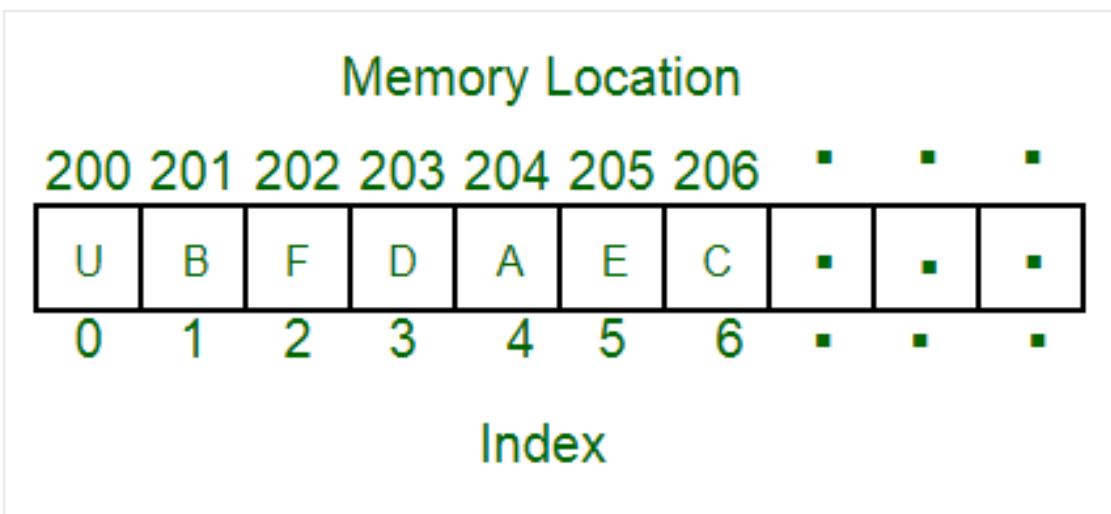
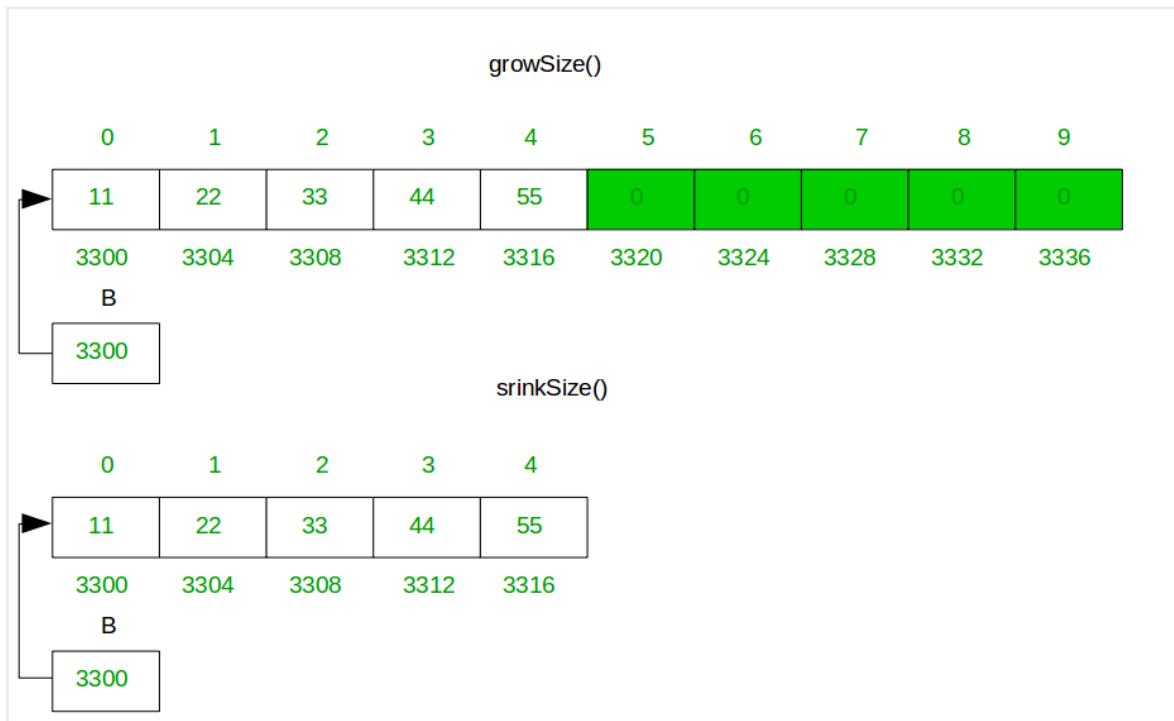


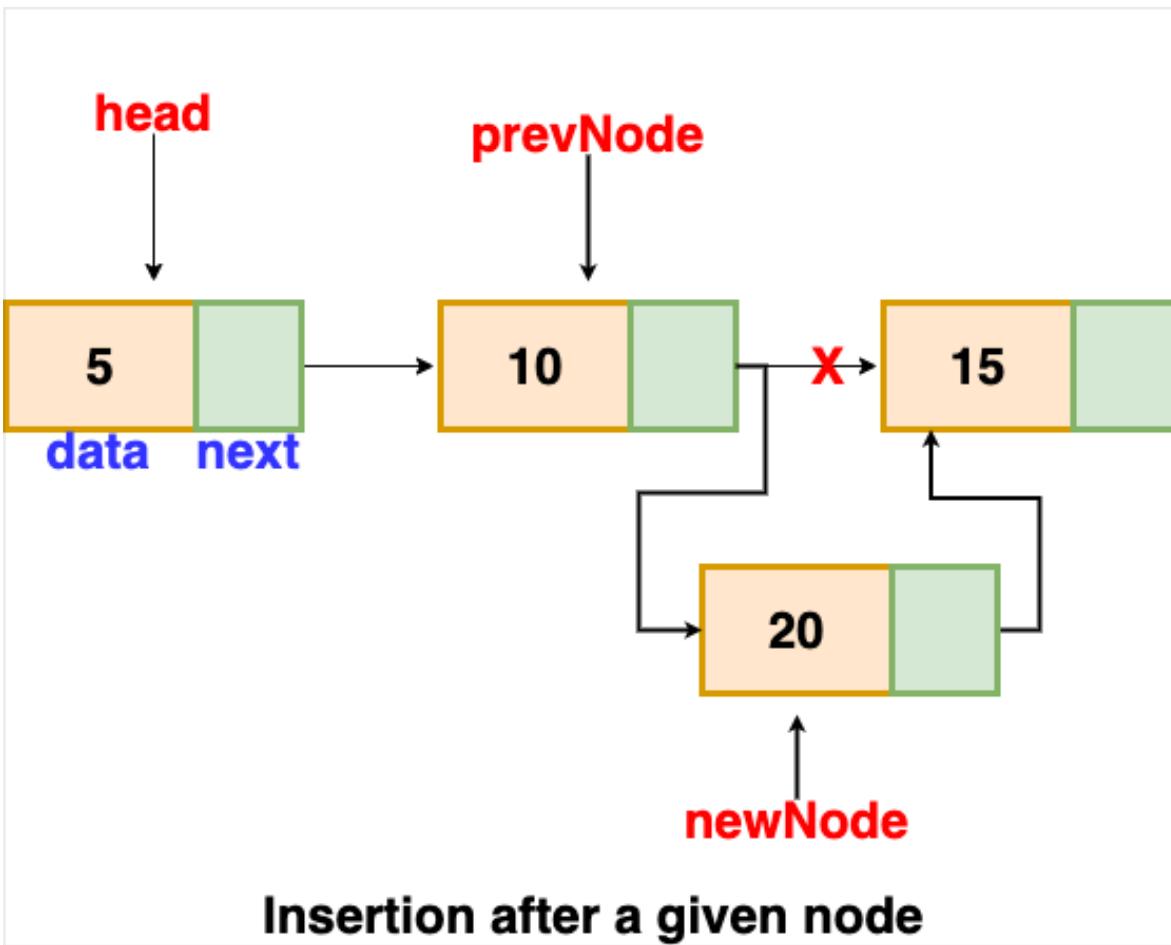
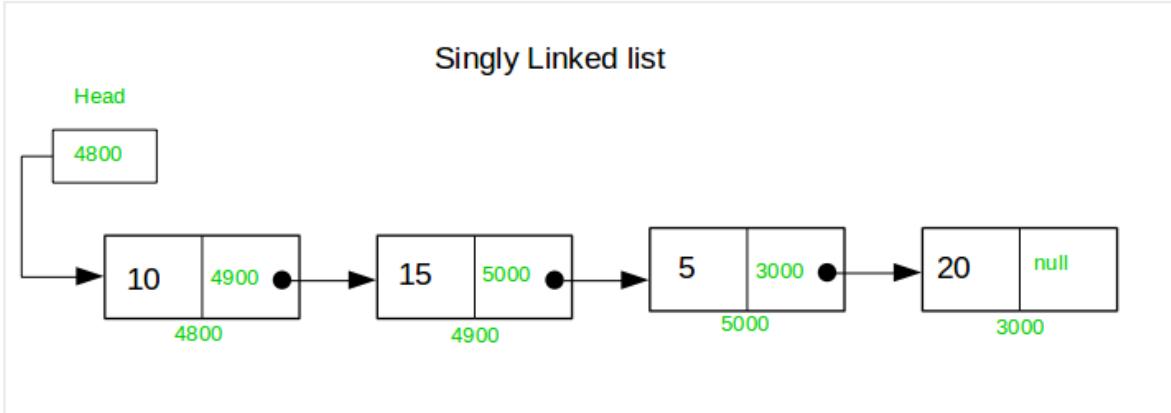
# Data Structure Design Questions for Coding Interviews

## 1. Arrays & Dynamic Arrays



- Design a **Dynamic Array** (like vector / ArrayList)
- Implement **insert**, **delete**, **resize**
- Implement **circular array**
- Design an array supporting:
  - O(1) getMin
  - O(1) getMax
- Implement **two stacks in one array**
- Implement **k stacks in one array**

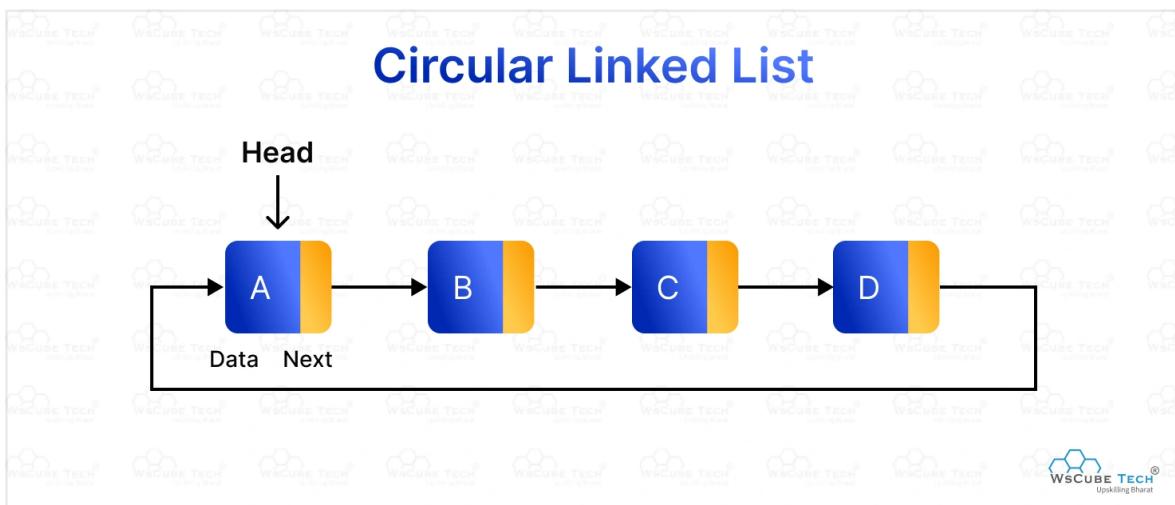
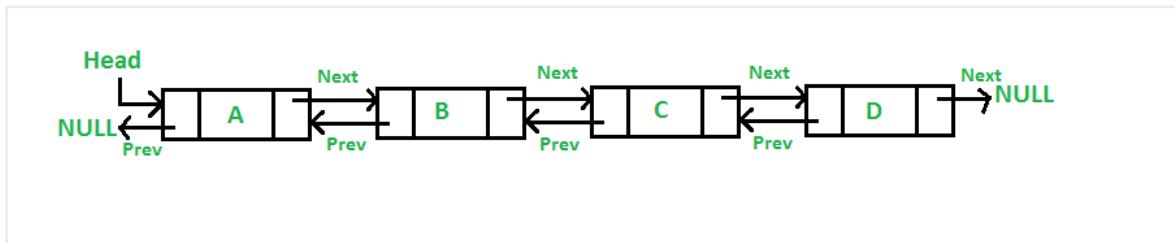
## 2. Singly Linked List



- Design a **Singly Linked List**
- Insert at:
  - head
  - tail
  - kth position
- Delete node:
  - by value
  - by position
- Reverse a linked list
- Find middle of linked list
- Detect cycle (Floyd)

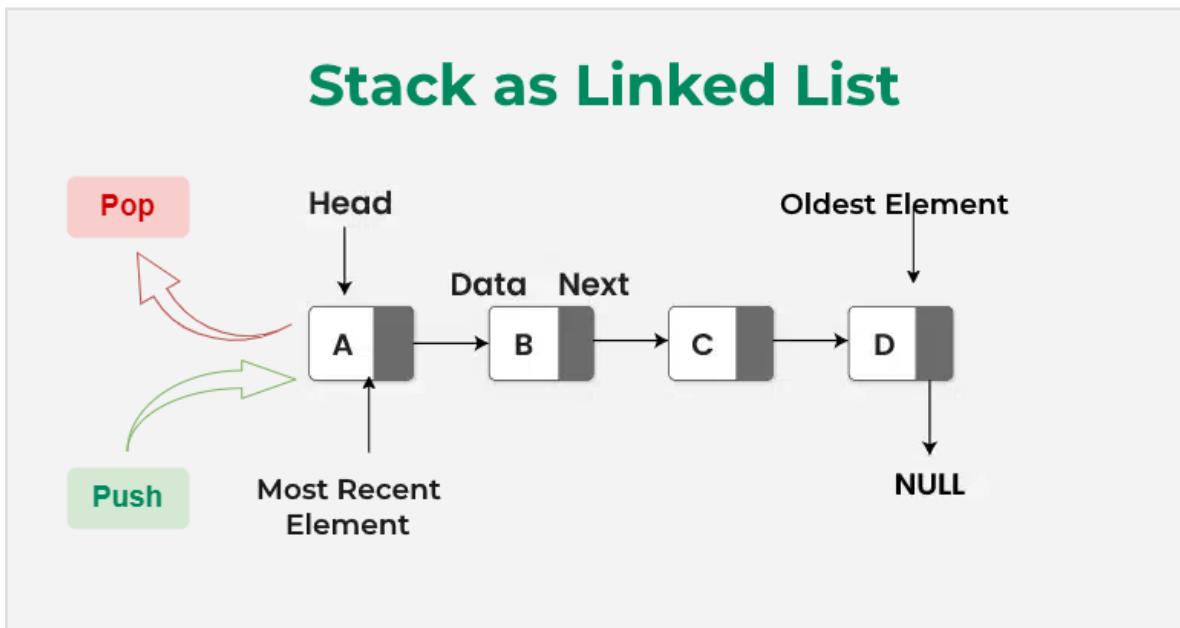
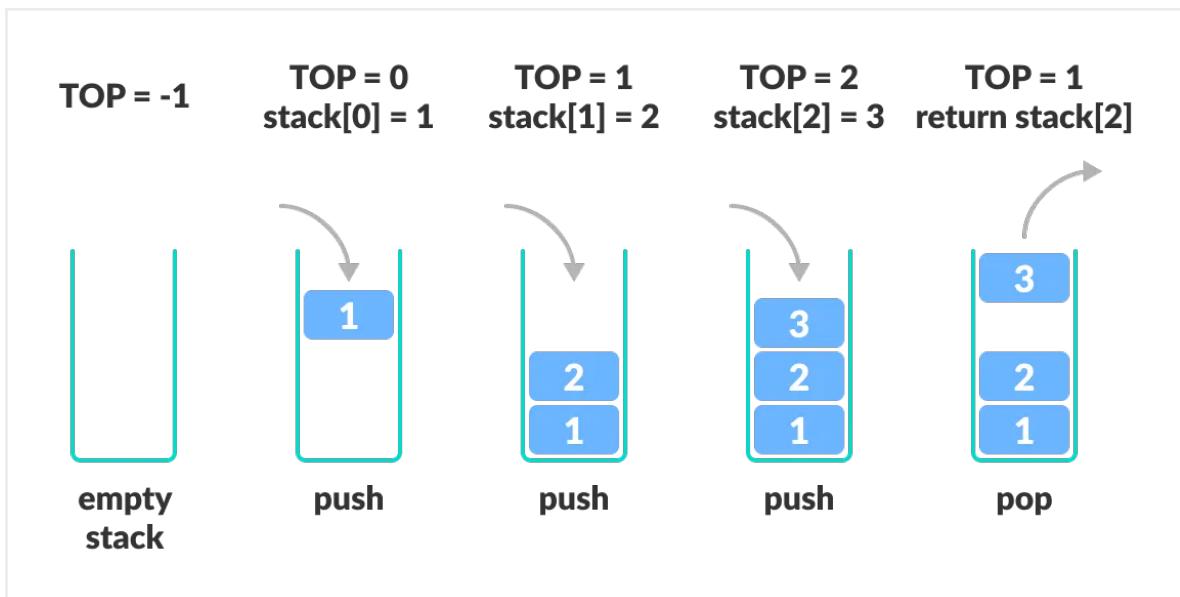
- Remove cycle
- Merge two sorted linked lists
- Design linked list with:
  - O(1) insertion
  - O(1) deletion
- Implement **LRU Cache** using linked list + hashmap

### 3. Doubly & Circular Linked List



- Design **Doubly Linked List**
- Design **Circular Linked List**
- Convert:
  - Singly → Doubly
- Insert/delete in circular list
- Implement browser **back/forward** navigation

### 4. Stack (Design Oriented)



- Implement stack using:
  - Array
  - Linked List
- Design **Min Stack**
- Design **Max Stack**
- Implement stack using:
  - Two queues
- Implement **k stacks**
- Design **Undo / Redo system**
- Evaluate expression using stack
- Implement **monotonic stack**

## 5. Queue & Deque



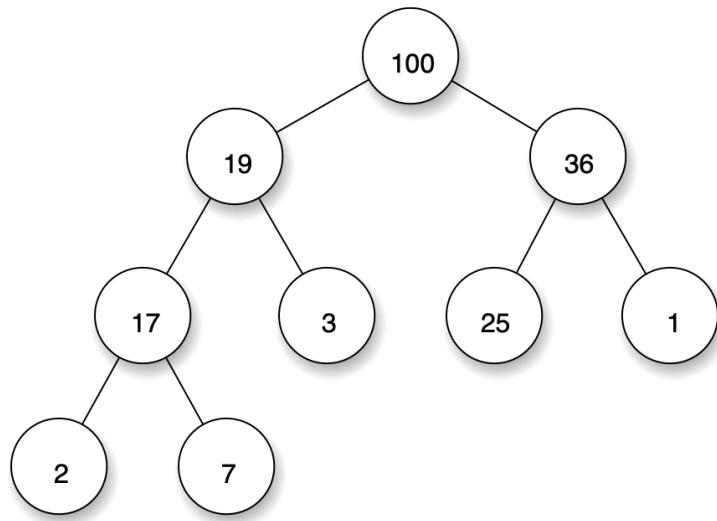
## Queue Data Structure



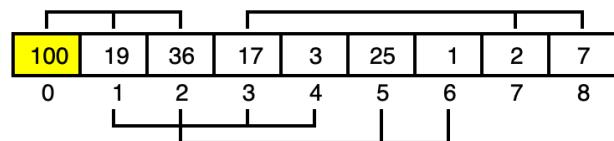
- Implement queue using:
  - Array
  - Linked List
- Design **Circular Queue**
- Design **Deque**
- Implement queue using:
  - Two stacks
- Design **Sliding Window Maximum**
- Design **Task Scheduler Queue**

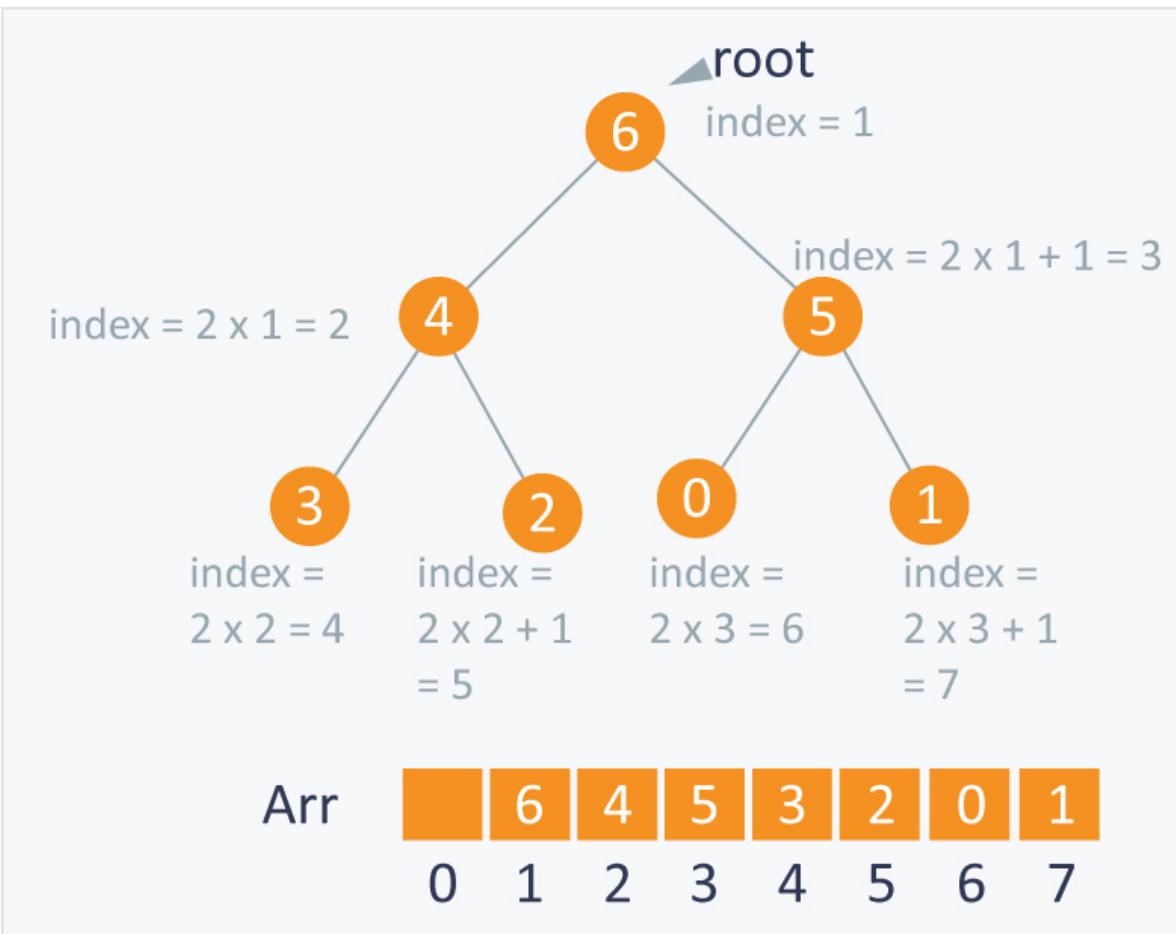
## 6. Priority Queue / Heap

## Tree representation



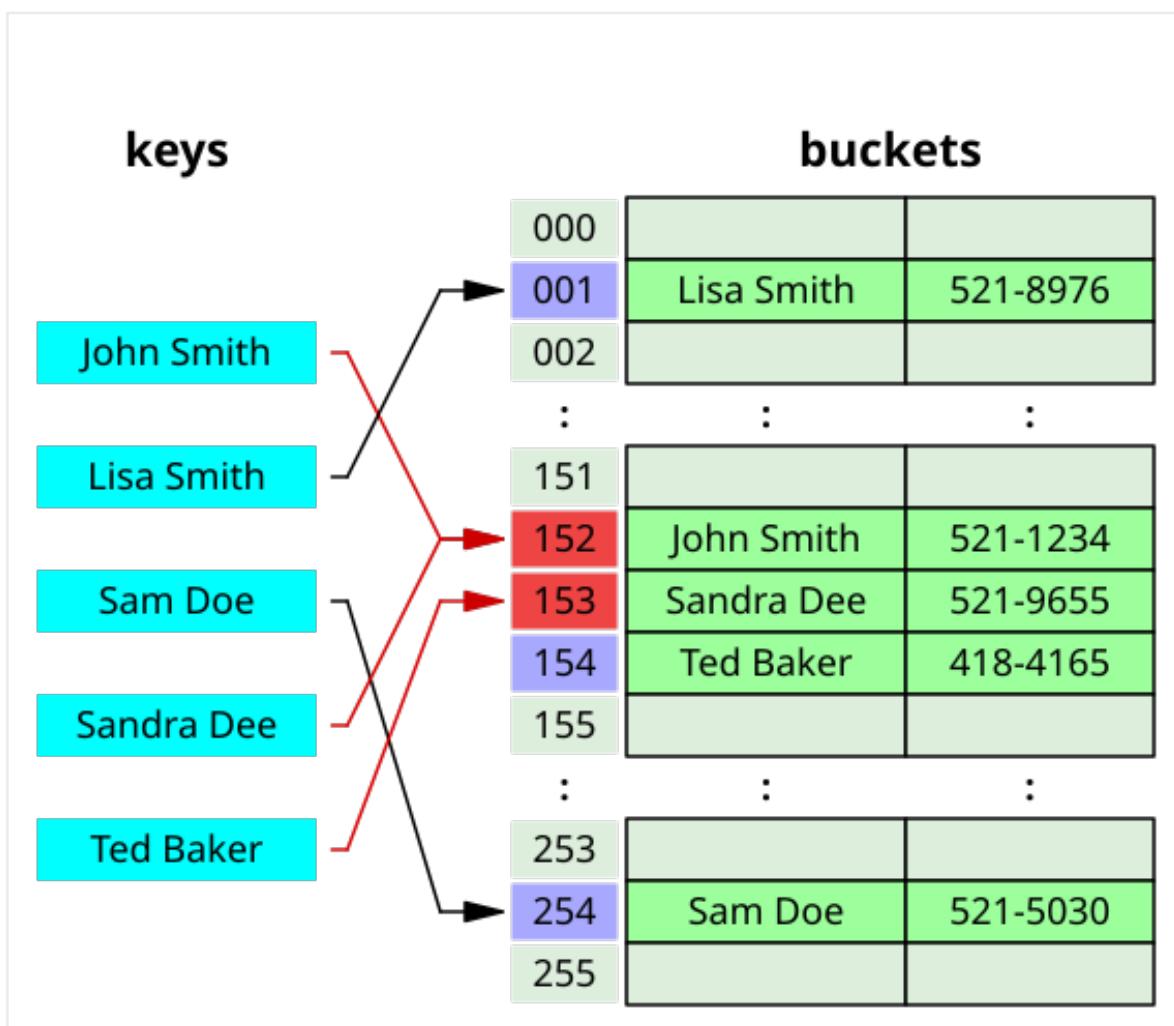
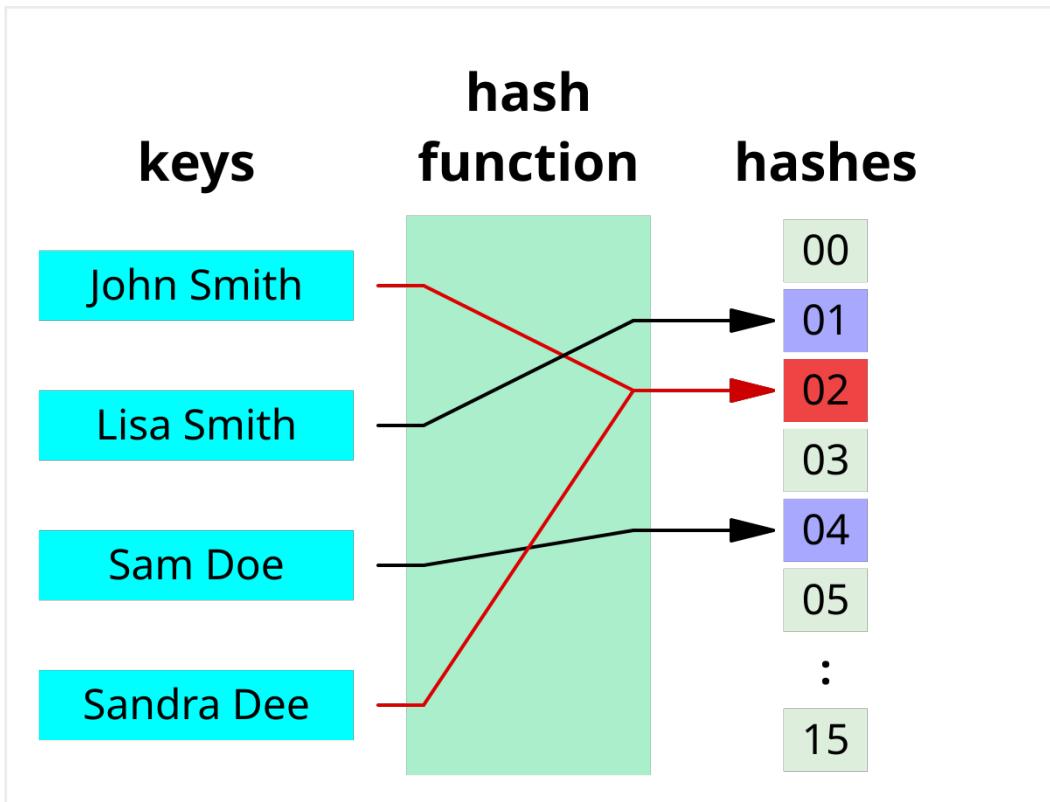
## Array representation





- Design **Min Heap**
- Design **Max Heap**
- Implement:
  - heapify
  - insert
  - delete
- Convert array → heap
- Implement **Priority Queue**
- Design **Median Finder**
- Design **Top K frequent elements**
- Design **Merge K sorted lists**

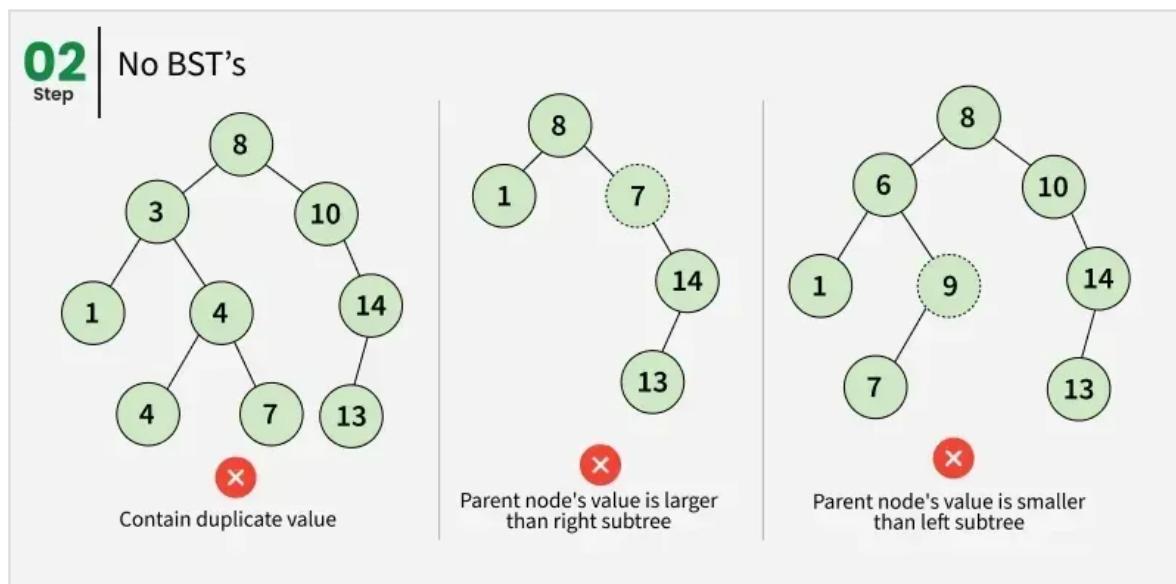
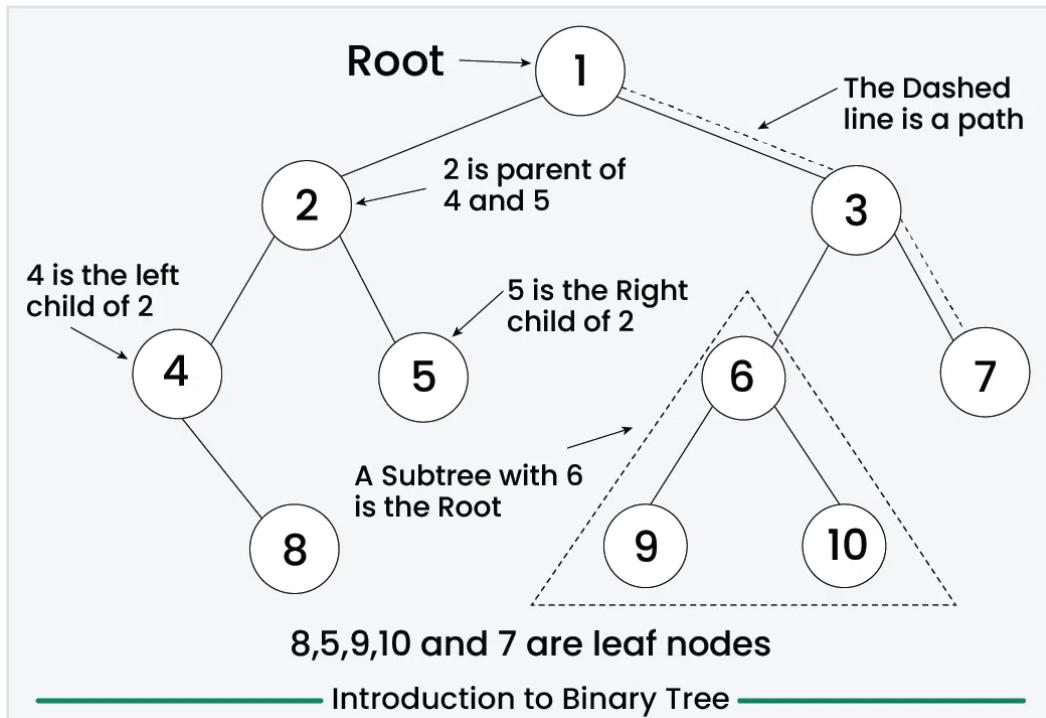
## 7. Hashing / HashMap



- Design **HashMap** from scratch
- Handle collisions:

- Chaining
- Open addressing
- Implement:
  - put
  - get
  - remove
- Design **HashSet**
- Design **LRU Cache**
- Design **LFU Cache**
- Design **Two Sum data structure**

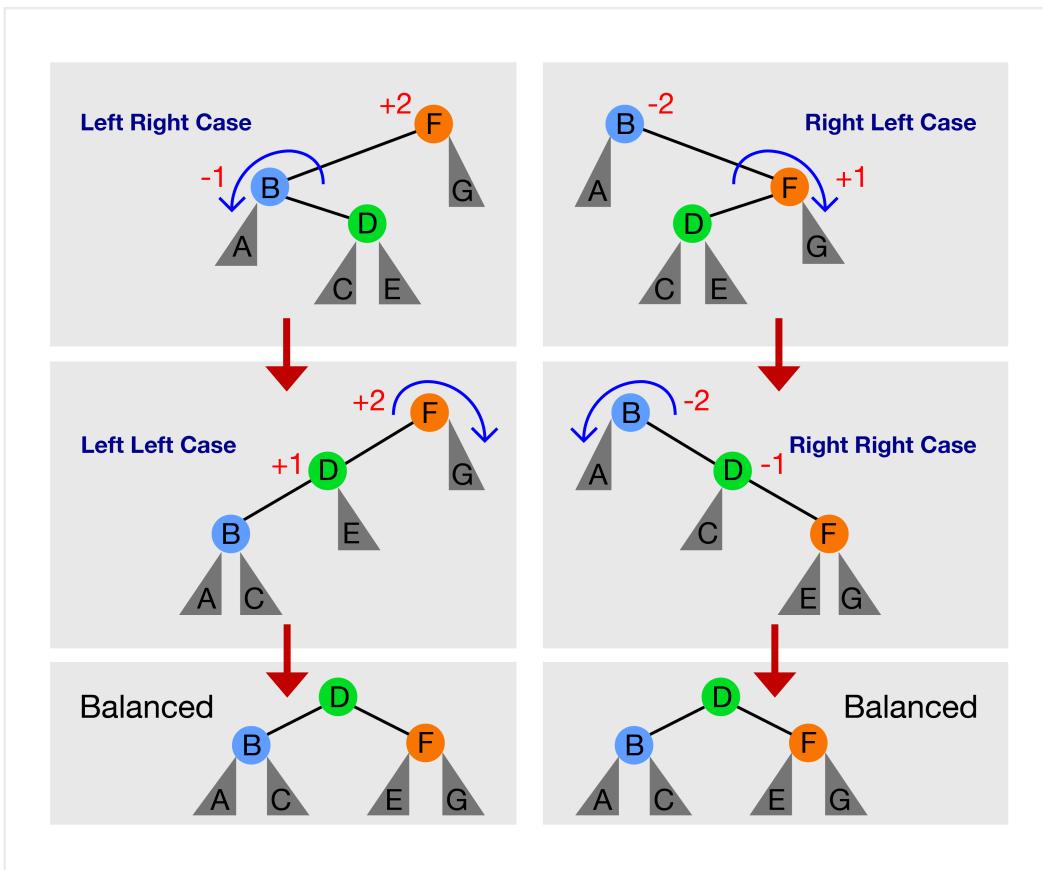
## 8. Trees (Core Design)



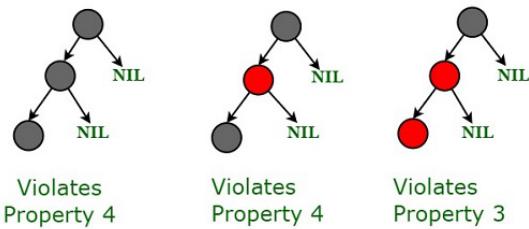
- Design **Binary Tree**

- Design **Binary Search Tree**
- Insert, delete, search
- Tree traversals:
  - inorder
  - preorder
  - postorder
- Level order traversal
- Find:
  - height
  - diameter
  - LCA
- Serialize & deserialize binary tree

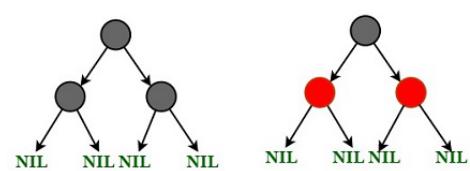
## 9. Balanced Trees



**Following are NOT possible  
3-noded Red-Black Trees**



**Following are possible  
Red-Black Trees with 3 nodes**



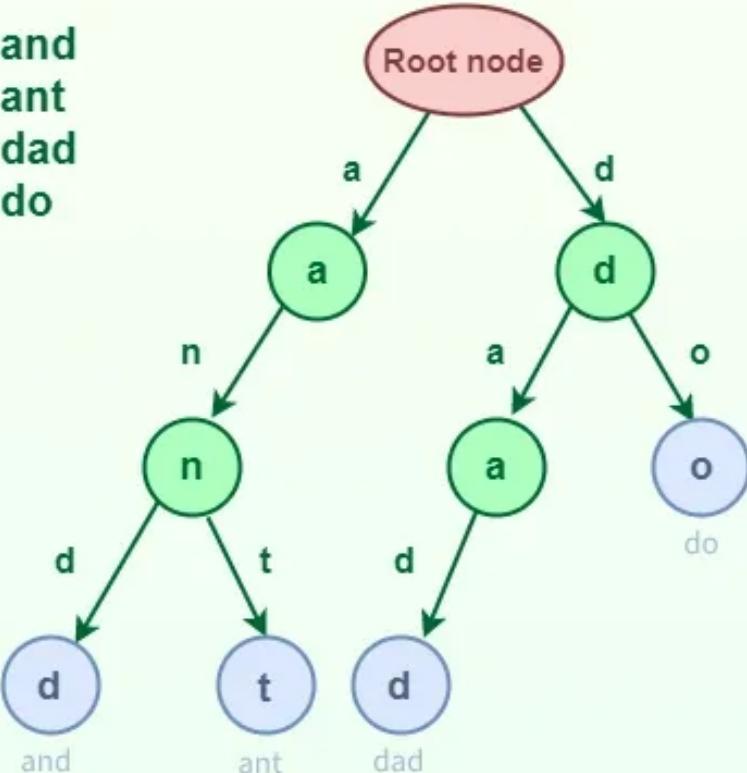
**All Possible Structure of a 3-noded Red-Black Tree**

- Design **AVL Tree**
- Perform rotations
- Design **Red-Black Tree** (conceptual + basic ops)
- Compare:
  - BST vs AVL vs RB Tree
- Design ordered map / set

## 10. Trie (Prefix Tree)

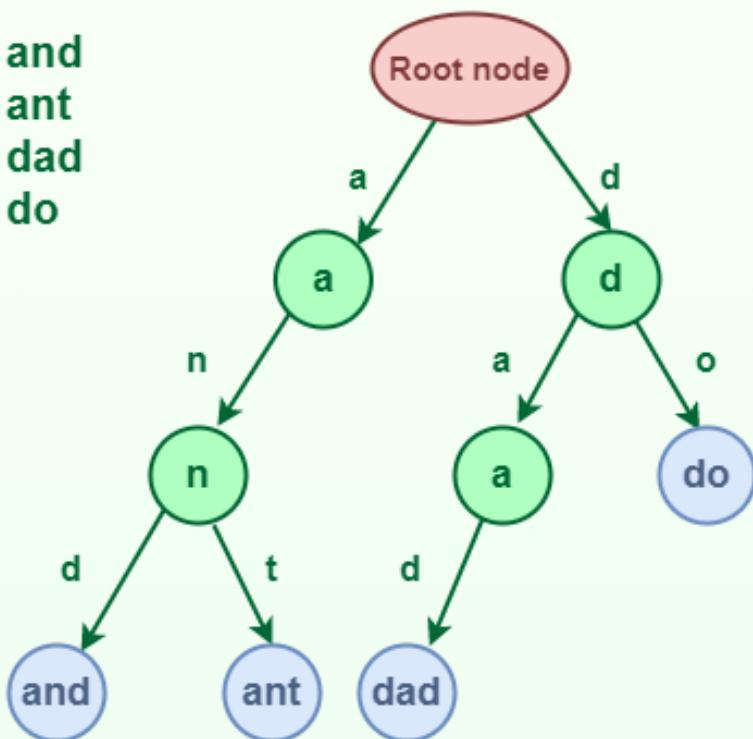
**Trie Data Structure**

- and
- ant
- dad
- do



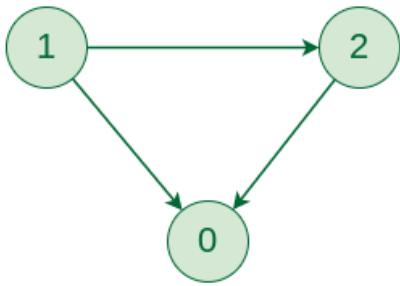
## Trie Data Structure

- and
- ant
- dad
- do

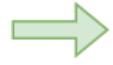


- Design **Trie**
- Insert, search, delete
- Implement:
  - auto-complete
  - prefix search
- Design **Word Dictionary**
- Longest prefix matching

## 11. Graph Design



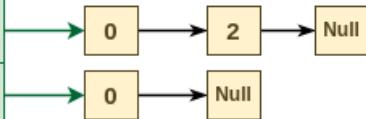
Directed Graph



Array

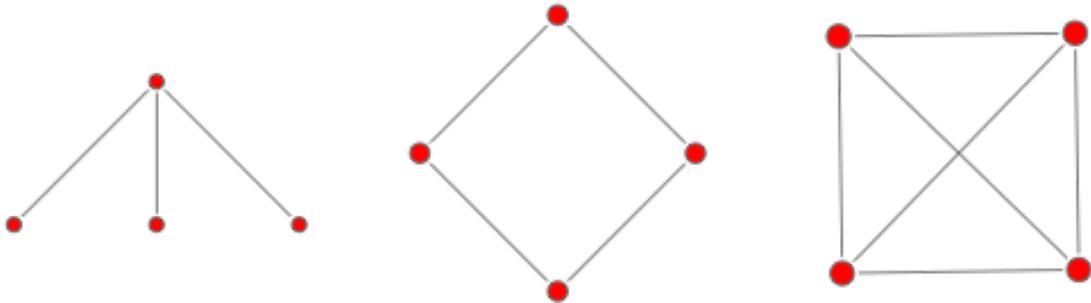
0
1
2

Linked List



Adjacency List

### Graph Representation of Directed graph to Adjacency List



$$\left( \begin{array}{ccc} & & \\ & & \\ & & \end{array} \right)$$

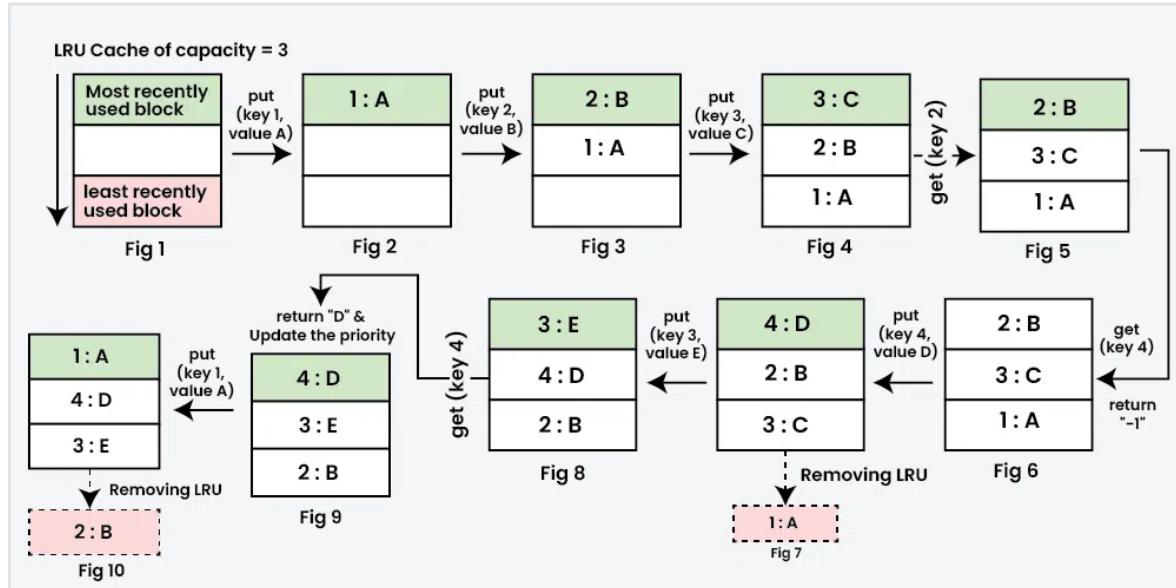
$$\left( \begin{array}{ccc} & & \\ & & \\ & & \end{array} \right)$$

$$\left( \begin{array}{ccc} & & \\ & & \\ & & \end{array} \right)$$

- Design graph using:
  - adjacency list
  - adjacency matrix
- BFS & DFS
- Detect cycle:
  - directed
  - undirected
- Design **Disjoint Set (Union Find)**

- Design **Topological Sort**
- Shortest path:
  - Dijkstra
  - BFS grid
- Design **Course Schedule system**

## 12. Advanced Design-Based DS



## 8 Data Structures That Power Your Databases

Types	Illustration	Use Case	Note
Skiplist		In-memory	Used in Redis
Hash Index		In-memory	Most common in-memory index solution
SS Table		Disk-based	Immutable data structure. Seldom used alone
		Memory + Disk	High write throughput. Disk compaction may impact performance
B-tree		Disk-based	Most popular database index implementation
Inverted index		Search document	Used in document search engine such as Lucene
Suffix tree		Search string	Used in string search, such as string suffix match
R-tree		Search multi-dimension shape	Such as the nearest neighbor

- Design **LRU Cache**
- Design **LFU Cache**
- Design **Rate Limiter**
- Design **File System**
- Design **In-Memory Database**
- Design **Key-Value Store**
- Design **Time-based Key Value Store**
- Design **Logger Rate Limiter**