## **Core Data Structures & Algorithms**

### **1. Trees & Graph Algorithms**

**Focus Areas:**

* **Lowest Common Ancestor (LCA)** - Binary lifting, Tarjan's offline algorithm
* **Tree traversals** - DFS, BFS, level-order
* **Dijkstra's Algorithm** - Single-source shortest path
* **Graph representations** - Adjacency lists, adjacency matrices
* **Path reconstruction** - Storing predecessors, backtracking paths

**Key Problems to Practice:**

* LCA in binary trees and general trees
* Shortest path in weighted graphs
* All-pairs shortest paths analysis
* Graph connectivity problems

### **2. Heap Data Structures**

**Priority Queues & Heaps:**

* **Min/Max heaps** - Implementation and operations
* **Priority queue operations** - Insert, extract, peek
* **Heap maintenance** - Heapify, bubble up/down
* **Custom comparators** - For complex objects
* **Lazy deletion** - Marking elements as deleted vs immediate removal

**Applications:**

* Task scheduling systems
* Resource allocation
* Top-K problems
* Median finding

### **3. Interval Processing**

**Key Concepts:**

* **Interval merging** - Overlapping intervals
* **Interval scheduling** - Non-overlapping maximum intervals
* **Sweep line algorithms** - Event-based processing
* **Conflict detection** - Resource booking systems

**Common Patterns:**

* Sort by start/end times
* Greedy selection strategies
* Event point processing

### **4. Concurrent Data Structures**

**Thread-Safety Concepts:**

* **Atomic operations** - Compare-and-swap, atomic counters
* **Lock-free data structures** - ConcurrentHashMap, ConcurrentSkipListMap
* **Synchronization primitives** - Mutexes, semaphores, read-write locks
* **Memory consistency** - Volatile variables, memory barriers

**Practical Applications:**

* Thread-safe counters
* Producer-consumer patterns
* Concurrent collections usage

## **Study Plan by Topic**

### **Week 1: Trees & Graphs**

* Implement LCA using binary lifting
* Practice Dijkstra's algorithm variants
* Solve tree traversal problems
* Work on path reconstruction techniques

### **Week 2: Heaps & Priority Queues**

* Implement min/max heap from scratch
* Practice priority queue scheduling problems
* Learn lazy deletion techniques
* Solve top-K and median problems

### **Week 3: Intervals & Scheduling**

* Master interval merging algorithms
* Practice scheduling optimization problems
* Learn sweep line techniques
* Solve calendar/booking system problems

### **Week 4: Concurrency & System Design**

* Study concurrent data structures
* Practice thread-safety patterns
* Learn about atomic operations
* Design thread-safe systems

## **Key Problem Types to Master**

### **Employee Directory (Tree + Concurrency)**

* Organizational hierarchy traversal
* Manager-employee relationships
* Concurrent updates to directory
* Efficient ancestor queries

### **Tennis Court Scheduler (Priority Queue)**

* Booking priority management
* Resource availability tracking
* Maintenance window scheduling
* Conflict resolution strategies

### **Commodity Tracker (Heap + Concurrency)**

* Real-time price tracking
* Thread-safe price updates
* Lazy deletion of outdated prices
* Efficient min/max price queries

### **Content Popularity (Concurrent Collections)**

* Real-time popularity tracking
* Thread-safe counter updates
* Efficient top-N content retrieval
* Sliding window popularity metrics

## **Implementation Tips for Interviews**

### **Code Quality Focus:**

* **Error handling** - Null checks, boundary conditions
* **Edge cases** - Empty inputs, single elements
* **Time/space complexity** - Always analyze and optimize
* **Clean code** - Readable variable names, modular functions

### **Concurrency Considerations:**

* **Race conditions** - Identify and prevent
* **Deadlock prevention** - Lock ordering, timeouts
* **Performance** - Minimize lock contention
* **Testing** - Concurrent execution scenarios

### **System Design Integration:**

* Think about scalability from the start
* Consider data consistency requirements
* Design for high throughput scenarios
* Plan for failure scenarios and recovery

## **Recommended Practice Resources**

1. **LeetCode** - Focus on medium/hard tree, heap, and interval problems
2. **System Design** - Practice designing schedulers and tracking systems
3. **Java Concurrency** - Study java.util.concurrent package

Practice explaining concurrent algorithms

The key to success is combining strong algorithmic skills with practical system design thinking, especially around concurrent and distributed systems. Make sure you can not only solve the algorithmic problem but also discuss how your solution would work in a real production environment with multiple threads and high load.

# Week 1: Trees & Graphs - Practice Problems

Here are specific problems organized by difficulty and concept for your Week 1 preparation:

## **Day 1-2: Tree Fundamentals & LCA**

### **Easy Problems**

1. **Binary Tree Level Order Traversal** (LeetCode 102)
   1. Foundation for tree traversal patterns
   2. Practice BFS implementation
2. **Lowest Common Ancestor of a Binary Tree** (LeetCode 236)
   1. Start with basic LCA in binary trees
   2. Understanding parent-child relationships

### **Medium Problems**

1. **Lowest Common Ancestor of a Binary Search Tree** (LeetCode 235)
   1. Leverage BST properties for optimization
   2. Compare with general tree approach
2. **Kth Ancestor of a Tree Node** (LeetCode 1483)
   1. Introduction to binary lifting technique
   2. Preprocessing for efficient queries

## **Day 3-4: Advanced Tree Algorithms**

### **Medium Problems**

1. **Binary Tree Maximum Path Sum** (LeetCode 124)
   1. Complex tree traversal with state management
   2. Understanding global vs local optimization
2. **Serialize and Deserialize Binary Tree** (LeetCode 297)
   1. Tree reconstruction techniques
   2. Important for distributed systems

### **Hard Problems**

1. **Binary Lifting for LCA** (Custom Implementation)
   1. Implement O(log n) LCA queries
   2. Preprocess tree with 2^k ancestors

## **Day 5-6: Graph Algorithms & Shortest Paths**

### **Medium Problems**

1. **Network Delay Time** (LeetCode 743)
   1. Direct application of Dijkstra's algorithm
   2. Single-source shortest path
2. **Path With Maximum Probability** (LeetCode 1514)
   1. Modified Dijkstra for probability maximization
   2. Understanding algorithm adaptations

### **Hard Problems**

1. **Cheapest Flights Within K Stops** (LeetCode 787)
   1. Constrained shortest path problem
   2. Bellman-Ford or modified Dijkstra

## **Day 7: Integration & Complex Problems**

### **Hard Problems**

1. **All Paths From Source Lead to Destination** (LeetCode 1059)
   1. Graph traversal with path validation
   2. Combines DFS with path analysis
2. **Shortest Path Visiting All Nodes** (LeetCode 847)
   1. TSP variant with BFS
   2. State compression techniques

## **Detailed Problem Breakdown**

### **Priority 1: Must Solve (Foundation)**

**Problem: Lowest Common Ancestor of a Binary Tree (LeetCode 236)**