Data Structures

<https://techdevguide.withgoogle.com/paths/data-structures-and-algorithms/>

When preparing for interviews at FAANG companies (Facebook, Amazon, Apple, Netflix, Google), it's crucial to have a solid understanding of various **data structures**. These form the backbone of the technical questions you'll face during coding interviews. Here's a comprehensive overview of the most important data structures you should be familiar with:

**1. Arrays and Strings**

* **Arrays** are the most fundamental data structure.
  + **Common Operations**: Access, Insert, Delete, Search, Traverse.
  + **Applications**: Sorting algorithms (e.g., Merge Sort, Quick Sort), dynamic arrays (like Python lists), sliding window problems, two-pointer techniques.
* **Strings**: Strings are essentially arrays of characters.
  + **Common Operations**: Searching, pattern matching (e.g., KMP algorithm), palindrome checking, substring search, anagram checking.

**2. Linked Lists**

* A **Linked List** is a sequence of elements where each element points to the next.
  + **Types**: Singly Linked List, Doubly Linked List, Circular Linked List.
  + **Operations**: Insertion at head, insertion at tail, deletion, reversal, detecting cycles (e.g., Floyd's Cycle-Finding Algorithm).
  + **Applications**: Efficient insertions/deletions in the middle of lists, representing polynomial expressions, LRU cache implementation.

**3. Stacks**

* A **Stack** follows a Last In First Out (LIFO) order.
  + **Operations**: Push, Pop, Peek (Top element), isEmpty.
  + **Applications**: Expression evaluation (infix to postfix conversion), undo/redo operations, parsing syntax (like in balanced parentheses problems), depth-first search (DFS).

**4. Queues**

* A **Queue** follows a First In First Out (FIFO) order.
  + **Types**: Regular Queue, Circular Queue, Priority Queue, Double-ended Queue (Deque).
  + **Operations**: Enqueue, Dequeue, Front, isEmpty.
  + **Applications**: BFS traversal, scheduling algorithms (e.g., CPU scheduling), task management, sliding window problems, level-order traversal in trees.

**5. Hash Tables (Hash Maps)**

* A **Hash Table** is a data structure that maps keys to values using a hash function.
  + **Common Operations**: Insert, Delete, Search, Update.
  + **Applications**: Counting frequencies, finding duplicates, implementing associative arrays (key-value pairs), caching (e.g., LRU cache), union/find operations in disjoint sets.

**6. Heaps**

* A **Heap** is a complete binary tree that satisfies the heap property:
  + **Types**: Max-Heap (root is the largest), Min-Heap (root is the smallest).
  + **Operations**: Insertion, Deletion (remove root), Peek (view root), Heapify (adjust heap).
  + **Applications**: Priority Queues (e.g., scheduling algorithms), Dijkstra’s shortest path algorithm, Huffman coding, dynamic median finding.

**7. Trees**

* **Binary Trees**: A tree where each node has at most two children.
  + **Binary Search Tree (BST)**: A binary tree where nodes follow the left < parent < right property.
  + **Balanced Trees**: AVL Tree, Red-Black Tree.
  + **Operations**: Insert, Delete, Search, Inorder/Preorder/Postorder traversal, balancing.
  + **Applications**: Searching (binary search), expression parsing (syntax trees), balanced search operations, file systems.
* **Tries (Prefix Tree)**:
  + A tree structure used for efficient storage and searching of strings.
  + **Operations**: Insert, Search, Prefix search, Delete.
  + **Applications**: Autocomplete, dictionary word search, prefix matching.

**8. Graphs**

* **Graphs** consist of vertices (nodes) and edges (connections between nodes).
  + **Types**: Directed Graphs, Undirected Graphs, Weighted Graphs, Unweighted Graphs, Bipartite Graphs, Complete Graphs.
  + **Operations**: Add Vertex, Add Edge, Remove Vertex, Remove Edge, Search (DFS/BFS), Topological Sort, Shortest Path (Dijkstra’s, Bellman-Ford), Minimum Spanning Tree (Prim’s, Kruskal’s), Cycle Detection.
  + **Applications**: Social networks, routing algorithms, dependency resolution, web page ranking (PageRank), network flow problems.

**9. Sets and Disjoint Set Union (Union-Find)**

* **Set** is a collection of unique elements.
  + **Operations**: Insert, Delete, Search, Union, Intersection, Difference.
  + **Applications**: Checking for duplicates, set operations, handling equivalence relations.
* **Disjoint Set Union (DSU)**: A specialized structure to handle the merging of sets and checking if two elements belong to the same set.
  + **Operations**: Union, Find (with path compression), Find with union by rank.
  + **Applications**: Kruskal's algorithm for MST, cycle detection in undirected graphs.

**10. Bit Manipulation**

* **Bitwise Operations** can be considered as a low-level data structure.
  + **Common Operations**: AND, OR, XOR, NOT, Left Shift, Right Shift.
  + **Applications**: Checking even/odd, finding unique elements in an array, optimizing space for boolean arrays (e.g., in bitmasking), efficient mathematical algorithms (like fast exponentiation).

**11. Dynamic Programming (DP) Arrays / Memoization**

* While not a data structure in itself, **DP** involves creating and maintaining arrays (or hash maps) to store intermediate results.
  + **Common Operations**: Memoization (storing computed values), Tabulation (bottom-up approach), transition from one state to another.
  + **Applications**: Optimization problems like Knapsack, Longest Common Subsequence, Matrix Chain Multiplication, Fibonacci sequence.

**12. Matrices**

* **2D Arrays** or **Matrices** are used for storing data in a grid.
  + **Operations**: Accessing elements, matrix multiplication, transpose, search, rotate matrix.
  + **Applications**: Image processing, dynamic programming (e.g., for problems like longest path, shortest path, etc.), graph representation (adjacency matrix).

**When preparing for FAANG interviews, here's how to focus your study:**

1. **Master the basics**: Arrays, linked lists, stacks, queues, and hash tables are foundational and often form the basis for other complex data structures and algorithms.
2. **Practice problem-solving**: It's crucial to not only understand the theory behind these data structures but also to implement them and solve algorithmic problems using them.
3. **Time and space complexity**: Understand the time and space complexities for operations on these data structures. For example, hash tables typically have O(1) average-time complexity for lookups, while binary search trees have O(log n) time complexity for search and insertion.
4. **Leverage online platforms**: Websites like **LeetCode**, **HackerRank**, **CodeSignal**, **CodeForces**, and **GeeksforGeeks** have extensive problem sets focusing on these data structures, which will help you practice solving algorithmic problems efficiently.