BOSSCODER ACADEMY

BEGINNER FRIENDLY

# DSA GUIDE

EVERYTHING YOU NEED TO KICKSTART YOUR DSA JOURNEY

Ultimate Guide to Building Strong DSA Fundamentals

# *Disclaimer*

EVERYONE LEARNS UNIQUELY.

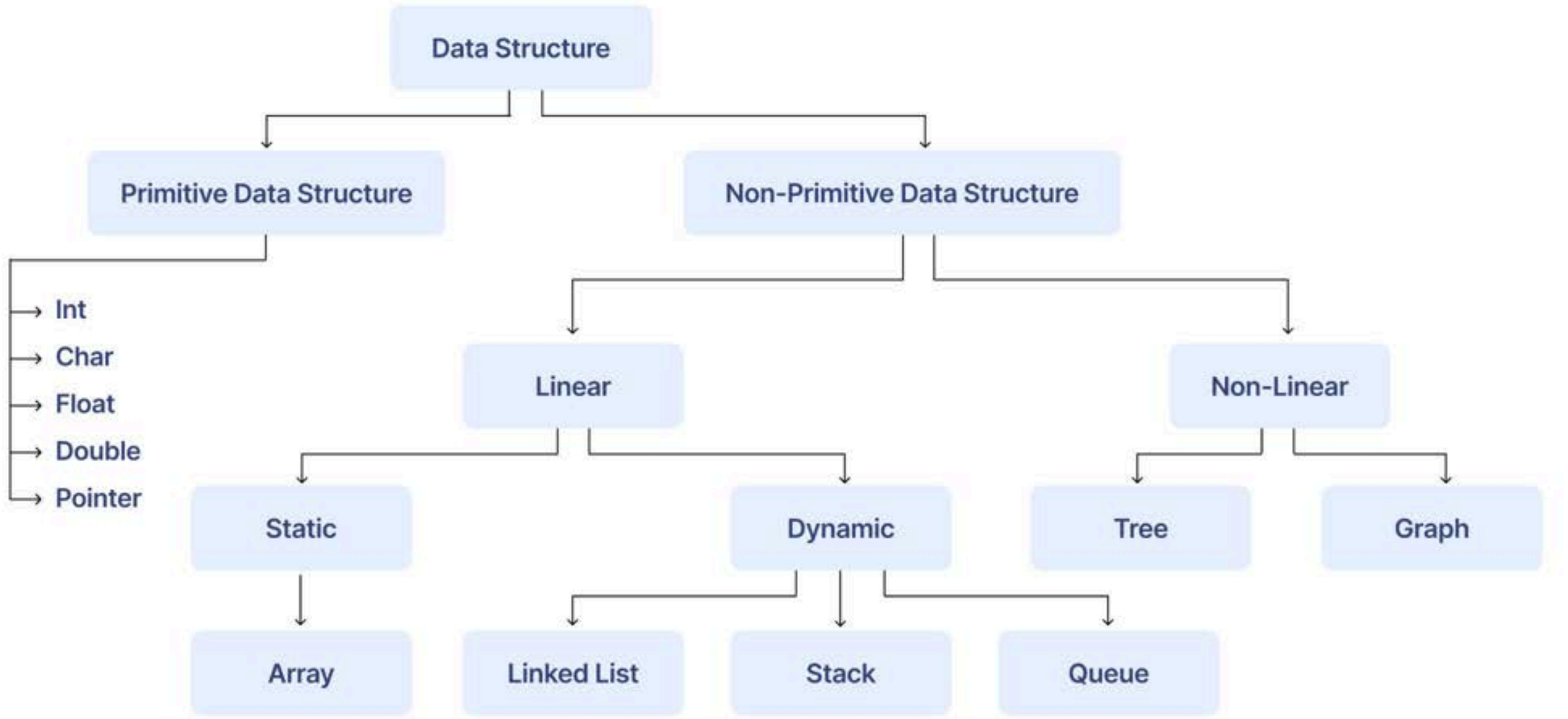
What matters is developing the problem solving ability
to solve new problems.

This Doc will help you with the same.

# Introduction —

If you're already comfortable with basic programming syntax in languages like C++, Java, or Python, it's time to build a strong foundation in **Data Structures and Algorithms (DSA)** — a crucial step for cracking coding interviews and solving real-world computational problems efficiently.

# Learn and Practice —




## 01 Time & Space Complexity

### WHAT TO LEARN

1. **Big O notation :**
    - Big O describes how the runtime or space requirement of an algorithm grows as the input size increases.
    - It gives an upper bound for performance, helping us compare algorithms.

## 2. Worst/Average/Best cases :

- **Worst Case:** Maximum steps algorithm will take (used in Big O)
- **Average Case:** Expected number of steps (average inputs)
- **Best Case:** Minimum steps (not commonly used for complexity)

**Practice Time/space complexity of basic operations (looping, recursion, searching)**

Practice

## 02 Array

An array is a data structure that can hold the elements of the same type. It cannot contain the elements of different types like integer with character.

The commonly used operation in an array is **insertion, deletion, traversing, searching.**

| # | Algorithm | Practice Link |
|---|---|---|
| 01 | Linear Search | Link |
| 02 | Binary Search | Link |
| 03 | Reverse an Array | Link |
| 04 | Find Min/Max in Array | Link |
| 05 | Basic Prefix Sum | Link |

## 03 Linked Lists

A linked list is a sequence of nodes where each node contains:

- Data (the value being stored)
- A pointer/reference to the next node in the sequence

Unlike arrays, **linked lists don't require contiguous memory allocation**, allowing for more flexible memory usage

### TYPES OF LINKED LIST:

- **Singly linked list:** Each node points to the next node, forming a one-way chain.
- **Doubly linked list:** Each node has pointers to both the next and previous nodes, allowing two-way traversal.
- **Circular linked list:** The last node points back to the first node, forming a closed loop.

| # | Algorithm | Practice Link |
|---|---|---|
| 01 | Insertion at Tail | Link |
| 02 | Delete given node | Link |
| 03 | Reverse a Linked List | Link |
| 04 | Merge Two sorted List | Link |

## 04 Stack & Queue

### STACK:

- A stack is a linear data structure that follows **LIFO (Last In First Out) order.**
- Operation: push, pop, top, isEmpty

| # | Algorithm | Practice Link |
|---|---|---|
| 01 | Implement stack using array | Link |
| 02 | Implement stack using Linked List | Link |
| 03 | Valid Parentheses | Link |
| 04 | Palindrome Linked List | Link |

### QUEUE:

- A queue is a linear data structure that follows **FIFO (First In First Out) order.**
- Operations: enqueue, dequeue, front, isEmpty

| # | Algorithm | Practice Link |
|---|---|---|
| 01 | Implement stack using array | Link |

| # | Algorithm | Practice Link |
|---|---|---|
| 02 | Implement stack using Linked List | Link |
| 03 | Number of Recent Calls | Link |

## 05 Strings

A string is a sequence of characters, they involve a small set of elements.

**For example:** Lowercase English alphabets → only 26 characters and ASCII characters → 256 characters

| # | Algorithm | Practice Link |
|---|---|---|
| 01 | Palindrome Check | Link |
| 02 | Basic String Manipulation | Link |
| 03 | Reverse String | Link |
| 04 | To Lower case | Link |

## 06 Searching

Searching is the process of finding an element in a given data structure (like an array or list).

### THERE ARE TWO COMMON TYPES:

1. **Linear Search:** Goes through each element one by one.
    - Time complexity: O(n)
    - Best when the data is unsorted or small in size.

| # | Algorithm | Practice Link |
|---|---|---|
| 01 | Searching in an Array | Link |

2. **Binary Search:** Repeatedly divides the array in half to find the element.
    - Works only on sorted arrays.
    - Time complexity: O(log n)

| # | Algorithm | Practice Link |
|---|---|---|
| 01 | Search in an Sorted Array | Link |
| 02 | Missing Number | Link |

# 07 Sorting

Sorting is the process of arranging elements in a particular order (ascending or descending).

It is used to improve the efficiency of searching, comparison, and data organization.

## TYPES OF SORTING:

- **Insertion Sort :** Builds the sorted array one element at a time by inserting elements into their correct position.
- **Merge Sort :** Divides the array into halves, recursively sorts them, and merges the sorted halves.
- **Quick Sort :** Picks a pivot, partitions the array around it, and recursively sorts the partitions.
- **Selection Sort :** Repeatedly selects the minimum element and places it at the beginning.

| # | Algorithm | Practice Link |
|---|---|---|
| 01 | Insertion Sort | Link |
| 02 | Merge Sort | Link |
| 03 | Quick Sort | Link |
| 04 | Selection Sort | Link |

## 08 Hashing

Hashing is a technique used to map data to a fixed-size value (called a hash) which helps in efficient data storage, retrieval, and lookup — especially in data structures like hash tables, dictionaries, and sets.

- Average-case time complexity: O(1) for insertion, deletion, and search.
- Used in applications like caching, dictionaries, sets, and databases.

A Hash collision occurs when two different keys produce the same hash value and are assigned to the same index in the hash table.

---

Since collisions are unavoidable in fixed-size tables, we need techniques to handle them.

1. **Separate Chaining:** Each index in the hash table contains a linked list of entries that hash to the same index. New elements are simply added to the list at that index.

2. **Open Addressing:** Instead of using a list, this method finds another empty slot in the table using a probing strategy.

   - **Linear Probing:** If a collision happens at index i, check i+1, then i+2, and so on.
   - **Quadratic Probing:** Probes in steps of squares: $i+1^2$, $i+2^2$, $i+3^2$, etc.
   - **Double Hashing:** Uses a second hash function to decide the step size for probing.

| # | Algorithm | Practice Link |
|---|---|---|
| 01 | Design HashMap | Link |
| 02 | Design HashSet | Link |

## 09 Recursion

Recursion is a programming technique where a function calls itself to solve smaller instances of the same problem.

- It is especially useful for problems that can be broken down into similar subproblems, such as tree traversal, factorial, and Fibonacci.
- Key Concept
  1. **Base Case:** The condition that stops recursion
  2. **Recursive Case:** The part where the function calls itself
  3. **Call Stack:** Keeps track of recursive function calls

| # | Algorithm | Practice Link |
|---|---|---|
| 01 | Factorial | Link |
| 02 | Power of Three | Link |

| # | Algorithm | Practice Link |
|---|---|---|
| 03 | Pow(x,n) | Link |
| 04 | Fibonacci Number | Link |
| 05 | Unique 3-digit Even Number | Link |

## 10 Trees

A tree is a hierarchical data structure made up of nodes, with one node designated as the root and all other nodes forming subtrees of the root.

It is a non-linear data structure, ideal for representing hierarchical relationships such as folders in a filesystem or organization structures.

### BINARY TREE TRAVERSALS (INORDER, PREORDER, POSTORDER)

| # | Algorithm | Practice Link |
|---|---|---|
| 01 | Inorder | Link |
| 02 | Preorder | Link |

| # | Algorithm | Practice Link |
|---|---|---|
| 03 | Postorder | Link |
| 04 | Same Tree | Link |
| 05 | Maximum depth of Binary Tree | Link |
| 06 | Path Sum | Link |

# 11 Heap

A Heap is a special tree-based data structure that satisfies the heap property:

- In a Max Heap, for every node i, the value of i is greater than or equal to its children.
- In a Min Heap, for every node i, the value of i is less than or equal to its children.
- It is a complete binary tree (all levels completely filled except possibly the last, filled from left to right)

## OPERATIONS ON HEAP :

1. **Heapify:** Rearranges a tree or array to maintain the heap property
2. **Insert:** Add a new element and adjust (heapify up)
2. **Delete:** Remove the root element and heapify down

| # | Algorithm | Practice Link |
|---|---|---|
| 01 | Operations on binary min heap | Link |

## HEAP SORT

It is a comparison-based sorting algorithm that uses a Max Heap (or Min Heap) to sort elements.

| # | Algorithm | Practice Link |
|---|---|---|
| 01 | Heap Sort | Link |
| 02 | Minimum Cost of ropes | Link |
| 03 | Kth Largest Element in a Stream | Link |
| 04 | Last Stone Weight | Link |

## 12 Greedy Algorithm

A Greedy Algorithm is a problem-solving approach that builds up a solution step by step, always choosing the option that looks best at the moment (locally optimal), hoping that this leads to a globally optimal solution.

| # | Algorithm | Practice Link |
|---|---|---|
| 01 | Longest Palindrome | Link |
| 02 | Assign Cookies | Link |

| # | Algorithm | Practice Link |
|---|---|---|
| 03 | Array Partition | Link |
| 04 | Can Place Flowers | Link |
| 05 | Lemonade Change | Link |

**Note:** Greedy algorithms don’t always guarantee the correct answer for every problem — only for those that satisfy the properties mentioned above. If a problem fails with greedy, dynamic programming might be the better fit.

## 13 Dynamic Programming

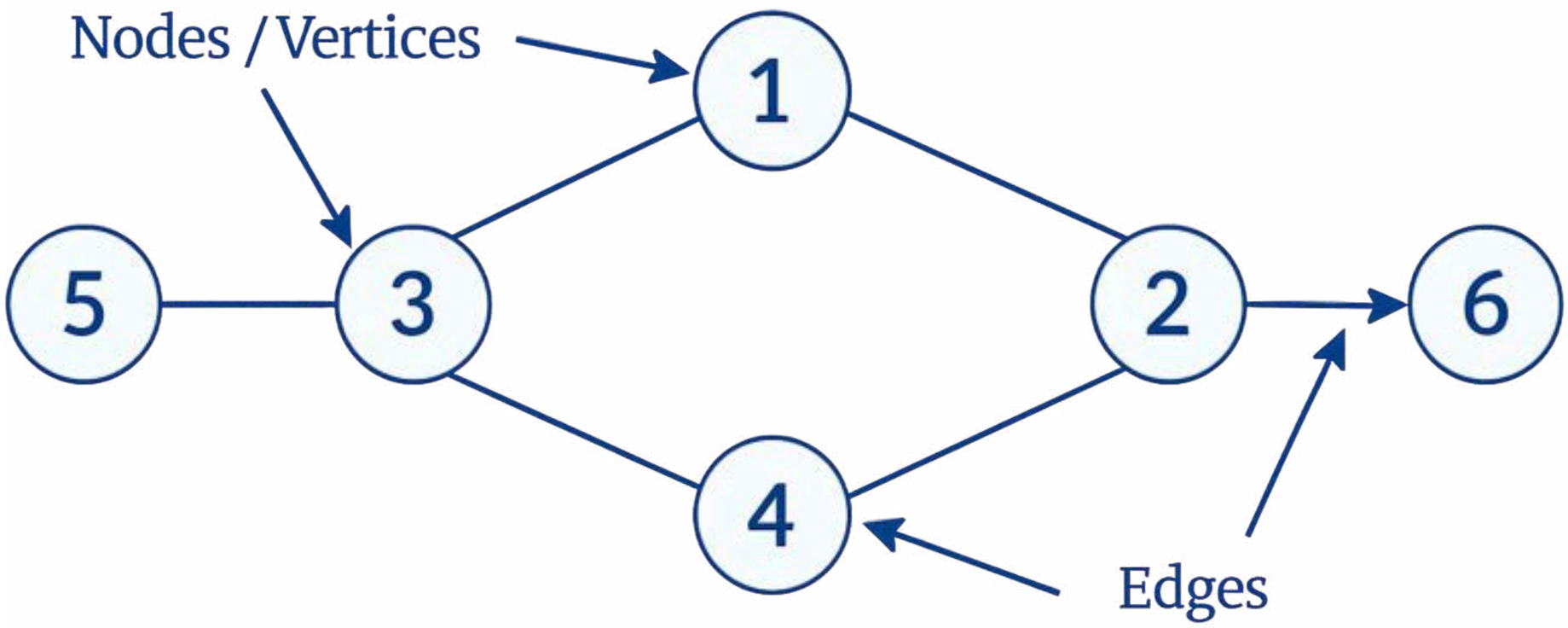
Dynamic Programming (DP) is a technique used to solve problems by breaking them down into smaller overlapping subproblems and storing their results to avoid redundant computations.

- It is especially useful in optimization problems and is based on two main strategies: **Memoization (Top-Down)** and **Tabulation (Bottom-Up).**
- DP helps in reducing the time complexity of problems that have **overlapping subproblems and optimal substructure.**

| # | Algorithm | Practice Link |
|---|---|---|
| 01 | Fibonacci (Memoization/ Tabulation) | Link |
| 02 | Climbing Stairs | Link |
| 03 | Pascal’s Triangle | Link |
| 04 | Best Time to Buy and Sell Stocks | Link |
| 05 | Counting Bits | Link |

# 14 Graph

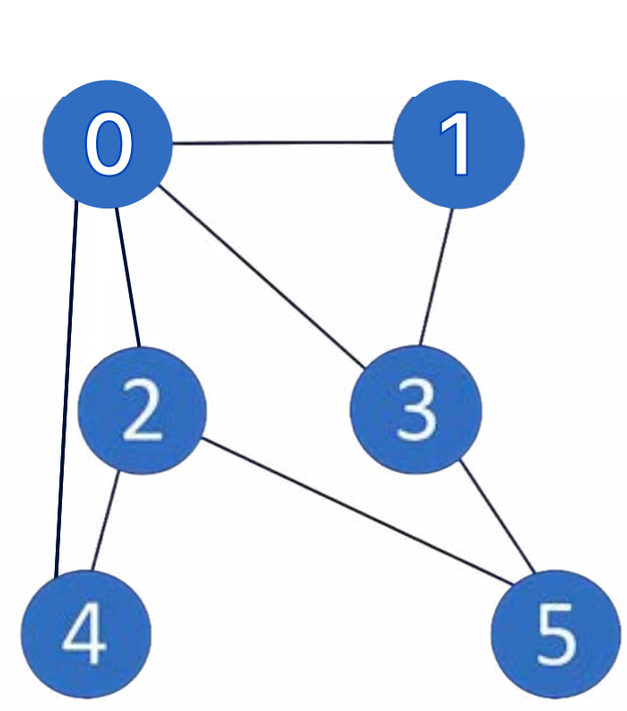
A Graph is a non-linear data structure consisting of nodes (also called vertices) and edges that connect them. It is used to model relationships and networks such as social connections, maps, and web page linking.




## NODE AND EDGES :

- A node represents an entity (e.g., a person, city, etc.)
- An edge represents a connection or relationship between nodes
- Graphs can be:
    1. Directed (edges have direction)
    2. Undirected (edges go both ways)

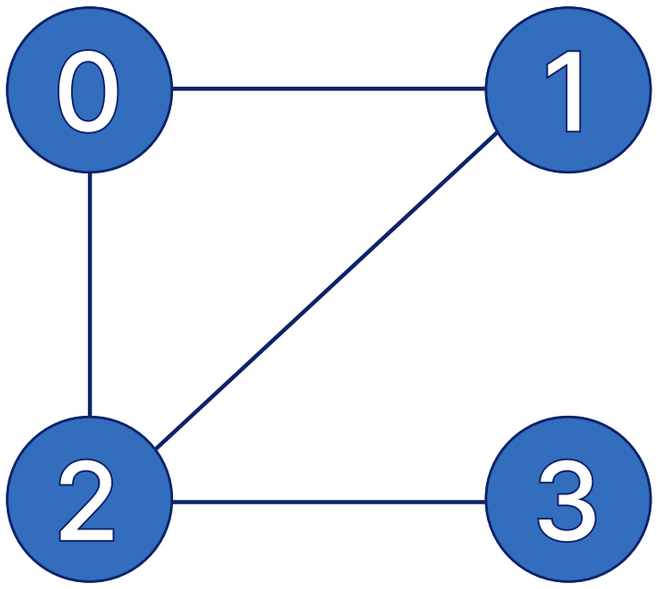
- A 2D array used to represent a graph
- matrix[i][j] = 1 means there is an edge from node i to node j.
- It takes more space (O(V²)).

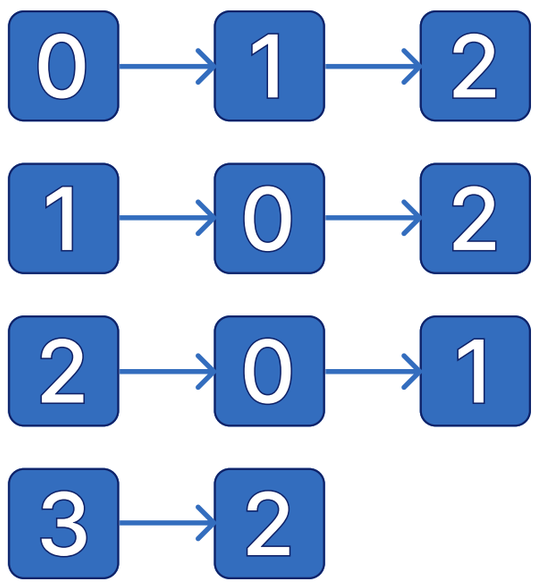



| | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 2 | 1 | 0 | 0 | 0 | 1 | 1 |
| 3 | 1 | 1 | 0 | 0 | 0 | 1 |
| 4 | 1 | 0 | 1 | 0 | 0 | 0 |
| 5 | 0 | 0 | 1 | 1 | 0 | 0 |

## ADJACENCY LIST :

- A node represents an entity (e.g., a person, city, etc.)
- An edge represents a connection or relationship between nodes



- Uses a list where each node stores a list of connected nodes
- More space-efficient than a matrix (O(V + E))
- Ideal for sparse graphs (i.e. where most nodes are not directly connected)

| # | Algorithm | Practice Link |
|---|---|---|
| 01 | Print adjacency list | Link |

## WEIGHTED GRAPH :

- Edges have weights or costs (e.g., distance, time, price)
- Used in algorithms like Dijkstra's or Prim's for shortest path problems

## GRAPH TRAVERSAL ALGORITHM :

### 1. BFS (Breadth-First Search)

- Explores all neighbours at the current depth before going deeper
- Implemented using a queue

| # | Algorithm | Practice Link |
|---|---|---|
| 01 | Rotting Oranges | Link |
| 02 | Find if Path Exists in Graph | Link |

## 2. DFS (Depth-First Search)

- Explores as far as possible along each branch before backtracking.
- Implemented using recursion or a stack.
- Useful for cycle detection, topological sort, etc.

| # | Algorithm | Practice Link |
|---|---|---|
| 01 | Directed Graph Cycle | Link |
| 02 | Number of Islands | Link |

BOSSCODER ACADEMY

# WHY BOSSCODER?

## 01 STRUCTURED INDUSTRY-VETTED CURRICULUM

Our curriculum covers everything you need to get become a skilled software engineer & get placed.

## 02 1:1 MENTORSHIP SESSIONS

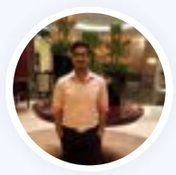
You are assigned a personal mentor currently working in Top product based companies.

## 03 2200+ ALUMNI PLACEMENT

**2200+** Alumni placed at Top Product-based companies.

## 04 24 LPA AVERAGE PACKAGE

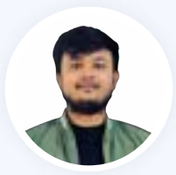
Our Average  Placement Package is **24 LPA** and highest is **98 LPA**



**Niranjan Bagade**
Software Engineer,
British Petroleum

**10 Years**
Experience

| | | |
|---|---|---|
| **NICE**<br>Software Eng. Specialist | Hike<br>**83%** | **British Petroleum**<br>Software Engineer |



**Dheeraj Barik**
Software Engineer 2,
Amazon

**2 Years**
Experience

| | | |
|---|---|---|
| **Infosys**<br>Software Engineer | Hike<br>**550%** | **Amazon**<br>SDE 2 |

**EXPLORE MORE**