Graphs in DSA: Beginner to Advanced Guide

Subtitle: Understand graphs, traversal techniques, and solve common coding interview problems efficiently.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app

Introduction

Graphs are versatile data structures used to model networks, social connections, and relationships. Mastering graphs is crucial for solving problems in competitive programming, algorithms, and interviews. This guide covers graph representation, traversal, and key problems.

Step 1: What is a Graph?

A graph is a collection of **nodes (vertices)** and **edges** connecting them.

- **Vertex (Node)** → fundamental unit
- **Edge** → connection between nodes
- **Directed Graph** → edges have direction
- **Undirected Graph** → edges have no direction
- **Weighted Graph** → edges carry values

Example:

1 -- 2

| |

3 -- 4

Step 2: Graph Representation

1. Adjacency Matrix

- 2D array, matrix[i][j] = 1 if edge exists
- Easy to implement, but uses O(V^2) space

2. Adjacency List

- Each node stores a list of connected nodes
- Efficient for sparse graphs

let graph = {

```
  0: [1, 2],
  1: [0, 2],
  2: [0, 1, 3],
  3: [2]
};
```

Step 3: Graph Traversal Techniques

1. Depth-First Search (DFS)

- Explore as far as possible along a branch

```
function dfs(node, visited=new Set()) {
  if (visited.has(node)) return;
  console.log(node);
  visited.add(node);
  for (let neighbor of graph[node]) {
    dfs(neighbor, visited);
  }
}
```

2. Breadth-First Search (BFS)

- Explore level by level

```
function bfs(start) {
  let queue = [start];
  let visited = new Set([start]);

  while (queue.length) {
    let node = queue.shift();
    console.log(node);
    for (let neighbor of graph[node]) {
      if (!visited.has(neighbor)) {
```

```
        visited.add(neighbor);
        queue.push(neighbor);
      }
    }
  }
}
```

Step 4: Important Graph Problems

1. Detect Cycle in Graph

- DFS for directed/undirected graphs
- Use visited + recursion stack

2. Shortest Path (Unweighted)

- BFS gives shortest distance in terms of edges

3. Topological Sort

- For directed acyclic graphs (DAG)
- Use DFS postorder or Kahn's algorithm

4. Connected Components

- Count disconnected subgraphs using DFS/BFS

Step 5: Weighted Graph Problems

1. Dijkstra's Algorithm

- Shortest path from source to all nodes
- Use min-priority queue

2. Bellman-Ford Algorithm

- Handles negative weights
- Detects negative cycles

3. Minimum Spanning Tree (MST)

- **Kruskal's Algorithm** → sort edges, union-find
- **Prim's Algorithm** → grow MST from a starting node

Step 6: Common Mistakes

- Forgetting **visited set** → infinite loops
- Confusing **DFS and BFS** applications
- Using adjacency matrix for large sparse graphs
- Ignoring **edge cases** like disconnected graphs

Step 7: Practice Plan for Graphs

Day 1–2:

- Graph representation + DFS/BFS → 5–7 problems

Day 3–4:

- Cycle detection + connected components → 5–7 problems

Day 5:

- Topological sort + shortest path → 5–7 problems

Day 6:

- Weighted graph + MST → 5–7 problems

Day 7:

- Mixed graph problems + revision → 10 problems

Mini Exercises

1. Count number of nodes in connected component
2. Find shortest path in unweighted graph
3. Detect cycle in directed graph
4. Implement BFS for level order traversal

Key Takeaways

- Graphs are **nodes + edges**, used for networks and relationships
- Master traversal: **DFS for depth, BFS for levels**
- Weighted graphs need specialized algorithms: **Dijkstra, Prim, Kruskal**
- Edge cases: disconnected nodes, cycles, negative weights
- Consistent practice improves intuition for competitive problems

Visit haas.dev for more DSA graph guides, practice problems, and interview preparation resources.

Website Name: haas.dev
Website Link: https://dev-roast-app.vercel.app