GRAPH EXERCISES

In this mini-workbook, we'll get you up and running with graphs.

In the guide, we will quickly highlight the key concepts that you need to know for your interview. If you're unfamiliar with any of these, I highly recommend taking some time to review them.

Then, I've provided you with a series of exercises to help you get up and running quickly. I recommend setting aside 2-3 days to work through these exercises. Simply start from the top and work through them as much as you have time.

Make sure to track the time you spend here in your Interview Prep Tracker. You can categorize this as "Study" time.

For these exercises, we have complete code samples and templates available here.

TABLE OF CONTENTS

Key Graph Terminology 2
Key Graph Patterns 2
Graphs Exercises 3

KEY GRAPH TERMINOLOGY

- **Directed Graph**
  A graph with connections that are directed. If Node A is connected to Node B, then that does not imply that Node B is connected to Node A

- **Undirected Graph**
  A graph where if Node A is connected to Node B, then Node B is connected to Node A

- **Weighted Graph**
  A graph where the edges between nodes are assigned a weight

- **Adjacency Matrix**
  A representation of a graph where an NxN boolean matrix represents the edges. matrix[i][j] represents an edge from Node i to Node j

- **Adjacency List**
  Similar to an adjacency matrix, except that we only track the actual edges and not all possible edges. For each node i, we keep a list of all of the nodes that i points to

KEY GRAPH PATTERNS

- **Accessing and Modifying Values**
  Fundamentally, a graph is just separate data points that are related to each other in some way. Therefore, when we implement a graph, we store two things:

  1. The data itself
  2. The relationships between the data

  With graphs, there are multiple ways that we can do this, the two most common of which are adjacency lists and adjacency matrices. You will want to be comfortable using both to represent graphs.

- **Graph Searching**
  With our emphasis on the connections between data, graph searching allows us to answer many questions about those connections. Are two data points connected? How closely are they connected? What is the path from one data point to the other?

  Many graph problems can be reduced to searching.

- **Topological Sorting**
  A topological sort allows us to order the nodes in our graph in terms of relative connectivity. This is useful in particular for ordering data based on dependencies (for example, which software packages depend on which other packages, so which need to be build first)

- **Implicit Graphs**
  Implicit graphs are graphs where we are not explicitly told what the connections are between our data. Rather, we determine that computationally in some way. Most real-world graph problems are implicit graphs. For example, if we are playing chess and the current state of the board is a node in the graph, we can find all the nodes that are connected to that node so long as we understand what all the possible valid chess moves we can make are.

- **[Bonus] Advanced Graph Algorithms**
  You DO NOT need to memorize any of these algorithms for your interview, but if you have extra time, implementing them is great graph practice (and included in the exercises below)
    - Dijkstra's Algorithm (Shortest Path in a Weighted Graph)
    - Prim's Algorithm / Kruskal's Algorithm (Find a Minimum Spanning Tree)

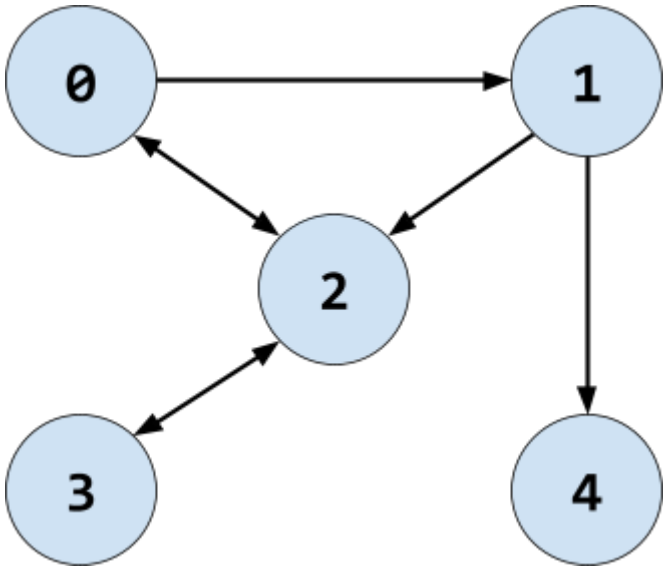
- Here is a great overview of all common graph algorithms you may need to know

GRAPH EXERCISES

Exercise Set #1 - Accessing and Modifying Values

1. Unlike all the other data structures that we've covered up to this point, graphs tend to be much less intuitve. Therefore, before we even get into any code let's make sure that we understand clearly how a graph is represented.

   Given the following graph, write out by hand the representation of the graph as:



   a. An adjacency matrix
   b. An adjacency list

2. Implement two classes to represent a graph:

   a. An adjacency matrix representation
   b. An adjacency list representation

   For each class, implement the following methods:
   a. Constructor
   b. `addEdge(int node1, int node2)` - Add an edge from `node1` to `node2`
   c. `removeEdge(int node1, int node2)` - Remove edge from `node1` to `node2`
   d. `neighbors(int node)` - Return all the neighbors of `node`

3. Implement a function to convert an adjacency matrix into an adjacency list and vice versa.

4. Write a function to clone a graph (Full Problem Definition)

Exercise Set #2 - Graph Searching

1. Write a function to determine whether a path exists between two nodes. You should implement this twice, once using an adjacency list and once using an adjacency matrix (Full Problem Definition).

   For the remaing exercises in this set, choose your preferred graph representation.

2. Write a function to find the length of the shortest path between two nodes (use BFS).

3. Write a function to find the shortest path between two nodes (use BFS).

4. Write a function to find ALL the paths between two nodes (use DFS).

Exercise Set #3 - Topological Sort

1. Course Scheduling (Full Problem Definition)

2. Course Scheduling II (Full Problem Definition)

3. Alien Dictionary (Full Problem Definition)

Exercise Set #4 - Implicit Graphs

1. Keys and Rooms (Full Problem Definition)

2. Evaluate Division (Full Problem Definition)

3. Sliding Puzzle (Full Problem Definition)