General Assembly

STUDY GUIDE

GRAPHS

What Is a Graph?

A graph is a powerful and flexible data structure that has been widely adopted in modern software engineering.

» Graphs are made up of nodes and edges.

- A **node** stores data (e.g., the name of a user on a social media platform).
- **Edges** represent relationships between nodes (e.g., "friendships" between users).

» The edges in a graph are a lot more flexible than the edges in a tree.

- Trees represent data that's hierarchical in nature: each node can be connected to a "parent node" above it and "child nodes" below it.
- Graphs represent relationships between data more generally: a node can have many edges to other nodes, and there is no "parent/child" relationship between nodes.

Directed and Undirected Graphs

Graphs can be either directed or undirected.

» In an **undirected graph**, edges are used to form a connection between two nodes, but these edges do not point from one node to another.

- Undirected graphs represent relationships that are always reciprocal.
- Example: a "friendship" between two users on Facebook.

» In a **directed graph**, edges have a direction, meaning that they point from one node to another.

- Directed graphs represent relationships that do not have to be reciprocal.
- Example: one user follows another user on Instagram.

Representing Graphs

There are also two ways to represent a graph in code: an **adjacency list** and an **adjacency matrix**.

» An **adjacency list** uses a collection of arrays.

- Each node has its own array, which lists all the other nodes it is connected to.
- An adjacency list is the more commonly used graph representation.
- An **adjacency matrix** is represented by a two-dimensional array.
- Each subarray is a node, and the values in the node represent edges to other nodes.
- In code, 1 represents an edge and a 0 represents a lack of an edge.

These two implementations differ in Big-O efficiency:

Operation	Adjacency List	Adjacency Matrix
Lookup	O(N)	O(1)
Add a node	O(1)	O(N^2)
Remove a node	O(N+E)	O(N^2)
Add an edge	O(1)	O(1)
Remove an edge	O(E)	O(1)

Operation	Adjacency List	Adjacency Matrix
Memory	O(N+E)	O(N^2)

Traversing a Graph

There are two common ways of traversing graphs:**breadth-first search** and **depth-first search**.
- **Breadth-first search:** Visiting each node that's connected to the root node before proceeding to the next level of nodes.
- Breadth-first search tries to stay as close to the starting point as possible before moving through the next parts of the graph and going into subsequent nodes

» **Depth-first search:** Following one chain of related nodes until you reach the bottom of the graph then returning to the root and starting over.

- Depth-first search tries to get as far away as possible from the starting point, visiting each node until it hits a dead end and then starting over.

The structure of your graph is the most important thing to consider when choosing a search method.
- If you know the value you're looking for is closer to the start of the graph, breadth-first is faster than depth-first.
- If the graph is very wide but not too deep, depth-first is more efficient than breadth-first.

Additional Resources

In a job interview, you might be asked to explain how a graph works conceptually or the types of problems it might solve.
- Popular graph interview problems and solutions
- Common interview questions about graphs that don't necessarily deal with coding
- Overview of Dijkstra's shortest path algorithm, another popular graph-related interview question
- Visualization of Dijkstra's shortest path algorithm