Problem: Implement depth-first search on a graph.

Constraints

- **Is the graph directed?**
  - **Yes**
- **Can we assume we already have Graph and Node classes?**
  - **Yes**
- **Can we assume this is a connected graph?**
  - **Yes**
- **Can we assume the inputs are valid?**
  - **Yes**
- **Can we assume this fits memory?**
  - **Yes**

Test Cases

Input:

- **add_edge(source, destination, weight)**
- **graph.add_edge(0, 1, 5)**
- **graph.add_edge(0, 4, 3)**
- **graph.add_edge(0, 5, 2)**
- **graph.add_edge(1, 3, 5)**
- **graph.add_edge(1, 4, 4)**
- **graph.add_edge(2, 1, 6)**
- **graph.add_edge(3, 2, 7)**

graph.add_edge(3, 4, 8)

Result:

- **Order of nodes visited: [0, 1, 3, 2, 4, 5]**

Algorithm

If we want to visit every node in a graph, we generally prefer depth-first search since it is simpler (no need to use a queue). For shortest path, we generally use breadth-first search.

- **Visit the current node and mark it visited**
- **Iterate through each adjacent node**
  - **If the node has not been visited, call dfs on it**

Complexity:

- **Time: O(V + E), where V = number of vertices and E = number of edges**
- **Space: O(V), for the recursion depth**