Answers to Self-Study Questions

Test Yourself #1

- graph (a) is a connected, complete, undirected graph
- graph (b) is a strongly-connected, complete, cyclic, directed graph
- graph (c) is a non-connected, not complete, acyclic, directed graph
- graph (d) is a weakly-connected (but not strongly-connected), not complete, acyclic, directed graph

Test Yourself #2

For a weighted graph, the edge weights could be stored in the nodes, with one weight for each outgoing edge (i.e., for each successor). So in addition to having a List of `Graphnode`s for the successors, there would be a List of values; the first value would be the label on the edge to the first successor, the second value would be the label on the edge to the second successor, etc.

Test Yourself #3

```
 public void resetAll() {
    for (Graphnode<T> n : nodes) {
        n.setVisited(false);
    }
}
```

Test Yourself #4

Question 1

Every topological order must start with A, and must include both C and D before E (but C can come before D or vice versa). Here are some possibilities:

```
A B C D E
A C D E B
A C B D E
A D C B E
```

Question 2

```
public void topOrderAll() {
    mark all nodes unvisited;
    int n = nodes.size();
    for each node k in the graph {
        if (node k is marked unvisited) {
            n = topNum(k, n);
        }
    }
}
```

Question 3

The solution to this question involves doing a modified depth-first search (following both outgoing and incoming edges from any node in the graph); the graph is connected if and only if all nodes are reached during that search.

```
public boolean isConnected( ) {
    if (nodes.size() == 0) {
        return true;
    }
```

```
        // do modified dfs from some node
        Graphnode<T> n = nodes.get(0);
        twoWayDfs(n);

        // graph IS connected if and only if all nodes were visited
        for (Graphnode<T>> n : nodes) {
            if (!n.getVisited()) {
                return false;
            }
        }
        return true;
    }

    private void twoWayDfs(Graphnode<T> n) {
        // do a dfs but follow predecessor edges as well as successors
        n.setVisited(true);
        for (Graphnode<T> m : n.getPredecessors()) {
            if (!m.getVisited()) {
                twoWayDfs(m);
            }
        }
        for (Graphnode<T> m : n.getSuccessors()) {
            if (!m.getVisited()) {
                twoWayDfs(m);
            }
        }
    }
```