Lab08-exercises

1. We have focused primarily on *time complexity* in this course, but when choosing data structures, space complexity is often as important of a constraint. Given an adjacency matrix, what is the 'space complexity' in Big-O. That is, given *n* nodes, how much space (i.e. memory) would I need to represent all of the relationships given. Explain your response.

   The space complexity for adjacency matrix is O(n^2), where n represents number of nodes. This is because to represent all relationships between nodes, we need a matrix with n rows and n columns. So, total memory needed is n * n, which is O(n^2).

2. Will it ever make sense for ROWS != COLUMNS in an adjacency matrix? That is, if we want to be able to model relationships between every node in a graph, must rows always equal the number of columns in an adjacency matrix? Explain why or why not.

   No, in adjacency matrix, rows must equal columns. This because, if we have n nodes, we need n rows and n columns to represent all possible connections between nodes. If ROWS not equal COLUMNS, some nodes' connections will be missing or extra nodes will be represented, which incorrect representation.

3. Can you run topological sort on a graph that is undirected?

   No, topological sort not possible on undirected graph. Topological sort used for directed acyclic graphs (DAGs) to find linear ordering of vertices, such that for every directed edge (u, v), vertex u comes before vertex v. In undirected graph, direction not there, so topological sort not applicable.

4. Can you run topological sort on a directed graph that has cycle?

   No, topological sort not possible on directed graph with cycle. Topological sort meant for directed acyclic graphs (DAGs). If graph has cycle, linear ordering of vertices not possible because cycle creates circular dependency.

5. For question number 4, how would you know you have a cycle in a graph? What algorithm or strategy could you use to detect the cycles? **Hint** we have already learned about this traversal.

   To detect cycle in graph, Depth-First Search (DFS) traversal used. During DFS, if visited node encountered again while traversing its children, cycle present in graph. By using DFS, we can detect cycles and know if topological sort possible or not.