Alan García

Written exercise:

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 is n^2, because forming a matrix requires n ROWS by n Columns.

   The only space that is arguably unused is the diagonal that goes across the matrix. If reflexivity was a universal value and didn't need to be included. Maybe we could find a way to make a slightly smaller matrix of size ROWS * (COLUMNS - 1). Just a thought.

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.

   Row and columns have the same size because we are pairing each node with every single 'other' node and itself. The size of n will always be n^2 because one side is multiplied by the other.

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

   It would be kind of pointless, we need a directed graph because we are trying to figure out which nodes go first and which ones go last.

   If we have undirected graphs, that means that each edge is symmetric which can make any node be a source or a leaf. Therefore, order might become something that could be defined in too many different ways depending on the implementation. It negates the purpose of having topological sort if the outcome can be completely different each time. With topological sort we are trying to minimize that as much as possible.

   The principles applied here can be used for a very specific set of problems, but I highly doubt we would still call it topological sort, so the answer is no.

   NOTE: we can create ADG from an undirected cyclic graph and then use this algorithm to solve.

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

   I tested this proposition using the tsort program. It's possible but it will throw an inaccurate order.

If we use topological sort on a graph that has cycles, we might get an arbitrary order because we are using an inverted dfs order. If this order has a cycle then the algorithm will alter the order and give us an inaccurate result.

5. For question number 4, how would you know if 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.

   As stated in question 4, we can use DFS to tag the visited nodes and if one visited node is visited again we can simply verify that this incident did or didn't happen.