Homework 4 Sample solutions

1. ```
   Initialize an empty adjacency list of length n for G^2
   For each u, initialize a boolean array Found[u][] (of length n) to false values
   for each edge (u, v)
     for each w adjacent to v
        if Found[u][w] is false and w is not equal to u
           add w to u's neighbor list in G^2
           set Found[u][w] = true
   ```

   The initialization before the loop is $O(n^2)$. The outer loop has m iterations and the inner loop has at most n iterations, since a given vertex has at most n neighbors. The remaining steps within the loop are all constant time, for an overall bound of $O(n^2 + mn)$, which simplifies to $O(mn)$ if the graph is connected.

2. Modify the algorithm in Section 3.3 of the text as follows, where instead of the boolean array `Discovered`, we define an array `Depth` to keep track of the level at which a given vertex is discovered.

   ```
   Set Depth[s] = 0 and Depth[u] = -1 ("undiscovered") for all other u
   Initialize L[0] to consist of the single element s
   Set ShortestCount[s] = 1 and ShortestCount[u] = 0 for all other u
   Set the layer counter i = 0
   While L[i] is not empty
     Initialize a new empty list L[i + 1]
     For each node u in L[i]
       Consider each edge (u, v) incident to u
         If Depth[v] < 0
           Set Depth[v] = i + 1
           ShortestCount[v] = ShortestCount[u]      (*)
           Add v to the list L[i + 1]
         else
           If Depth[v] is i + 1
             ShortestCount[v] += ShortestCount[u]    (**)
     Increment the layer counter i by one
   ```

   We know that if a node v is discovered at level i + 1 during a BFS, then a shortest path from s to v has length i + 1. It also follows that if node u is the predecessor of v on any shortest path, then u is at level i (otherwise, the path from s to u plus the edge from u to v would have length i + 2). Therefore the number of shortest paths from s to any node v at level i + 1 is equal to
   ```

$$\texttt{ShortestCount[u]} + \sum_{w \in L[i], (w,v) \in E} \texttt{ShortestCount[w]}$$

where u is the parent of v in the BFS tree, and the sum is taken over all w at level i that have an edge to v. We can prove by induction that the algorithm is correct:

- *Base step:* In the first iteration, `ShortestCount[v]` is set to 1 for each neighbor v of s, which is the correct value.
- *Induction step:* Let $k \geq 1$ and assume that `ShortestCount[u]` has the correct value for every vertex at level k. Let v be any vertex at level $k+1$. When v is first discovered, line (*) sets `ShortestCount[v]` to be the number of shortest paths to v's parent node. For every other node u at level k, if it has an edge to v, then line (**) adds the shortest count for u to the total for v, as required. Since v is an arbitrary node at level $k+1$, this shows that every node at level $k+1$ has the correct value for `ShortestCount`.

We conclude, by the principle of mathematical induction, that for every level k, the value of `ShortestCount[v]` is correct for every v at level k.

3. - We prove the contrapositive: if every node in a directed graph G has at least one outgoing edge, then G must have a cycle. Let G be a directed graph in which every node has at least one outgoing edge. Choose any node `current` and carry out the following steps $n+1$ times:

   ```
   Add current to path P
   Find an outgoing edge (current, v)
   Let current = v
   ```

   Since every node has at least one outgoing edge, the steps above construct a path P of length $n+1$. Since the graph has only n nodes, some node w must appear twice in P, forming a cycle.

   - This is similar to the algorithm in the text, but in reverse: the algorithm works by successively finding a node with no outgoing edges, prepending it to the topological ordering, and deleting the node from the graph.

   ```
   Initialize {\tt OutgoingCount[v]} to be the number of outgoing edges from v
   Initialize {\tt Incoming[v]} to be a list of all of v's incoming edges
   Add all nodes with no outgoing edges to a queue S.
   While S is not empty
     Remove the next element v from S
     Insert v at the beginning of the topological ordering
     For each incoming edge (u, v)
       decrement OutgoingCount[u]
       if OutgoingCount[u] = 0
         add u to S
   ```

4. Suppose that G' has a cycle $C = T_1, T_2, \ldots T_k$, where each T_i is a strongly connected component (SCC) of G. Since a cycle must include at least two distinct elements, we may arrange

our notation so that T_1 and T_k are distinct SCCs. By the definition of G', for each pair T_i, T_{i+1} in C there is an edge $\langle u_i, u_{i+1} \rangle$ in G with $u_i \in T_i$ and $u_{i+1} \in T_{i+1}$, and there is an edge $\langle v, w \rangle$ in G with $v \in T_k$ and $w \in T_1$. Let x be any vertex in T_1 and let y be any vertex in T_k. Since T_1 is an SCC, there is a path from x to u_1; likewise there is a path from u_k to y in T_k, and so there exists a path in G from x through $u_1, u_2, \ldots, u_k$ to y. On the other hand, since T_k is an SCC there is a path from y to v in T_k and likewise a path from w to x in T_1, forming a path from y to x in G. Since x and y were arbitrary nodes in T_1 and T_k, respectively, we have shown that all nodes in T_1 and T_k are in fact within the same SCC, contradicting the fact that T_1 and T_k are distinct nodes in G'. Therefore we conclude that G' cannot have a cycle. Since G' is directed by definition, this shows that G' is a DAG.