1. Consider the following sequence of union commands on the set of elements \{0, \ldots, 12\}:
   \begin{align*}
   &\text{union}(\text{find}(1), \text{find}(2)), \\
   &\text{union}(\text{find}(3), \text{find}(4)), \\
   &\text{union}(\text{find}(1), \text{find}(7)), \\
   &\text{union}(\text{find}(3), \text{find}(5)), \\
   &\text{union}(\text{find}(8), \text{find}(9)), \\
   &\text{union}(\text{find}(1), \text{find}(10)), \\
   &\text{union}(\text{find}(3), \text{find}(10)), \\
   &\text{union}(\text{find}(3), \text{find}(11)), \\
   &\text{union}(\text{find}(3), \text{find}(12)).
   \end{align*}

   Show the result when the unions are performed
   - using the naïve union method (left root becomes the child of the right root)
   - using union-by-size
   - using union-by-height

2. For the tree(s) obtained in the first path of the previous question -- where unions were done using naïve method -- show the result of performing path compression on the deepest node.

3. Consider the random maze generation algorithm given in Weiss (refer to the text for more details):
   - Start with each cell in the grid in its own set, all interior walls in place
   - Repeat {
     - Choose an interior wall at random
     - If the two cells (call them x and y) adjacent to this wall are in different equivalence classes, remove the wall and perform a union on x and y
   }

   Explain the following:
o What equivalence relation on cells is being dynamically built in this algorithm?

o Why do we not stop performing unions once the beginning cell and the end cell are in the same equivalence class?

o How does the algorithm guarantee that there will be only one path from the beginning cell to the end cell?

o Exactly how many unions are performed? Explain your answer.

4. Perform a topological sort on the directed graph shown below and list all the topological order of elements.

5. For the directed graph given below:
Find the shortest path from vertex A to all other vertices in the graph (indicate vertices which are unreachable).

Find the shortest unweighted path from vertex A to all other vertices.

Write down adjacency lists for the directed graph; use the convention that within a list, vertices appear in alphabetical order. For this choice of adjacency list, draw in the depth-first spanning tree (or forest).

Find the minimum spanning tree for the graph (disregard the direction of the edges) using Prim's algorithm. Is this minimum spanning tree unique? Explain.

Programming Problem:

For this problem, you will implement a version of Dijkstra's algorithm with slightly enhanced "book-keeping" so that if there are several minimum distance paths from a source vertex to a destination vertex in a weighted directed graph, you will return the one that has the fewest number of edges.

Recall that Dijkstra's algorithm, as described in the text, allows us to solve the single-source shortest weighted path problem: for a fixed source vertex \( s \), we can calculate the shortest distance \( d_v \) from \( s \) to \( v \) for every vertex \( v \) in the graph. The pseudocode given in Weiss also describes "book-keeping" to allow us to reconstruct one path whose distance is equal to the minimum distance \( d_v \).

Observe, however, that there can be several shortest distance paths from the source vertex \( s \) to the destination \( v \). For example, consider the graph consisting of the following set of weighted directed edges, given by triples (from, to, cost):

NYC LosAngeles 400
NYC SanFrancisco 700
NYC SanDiego 600
LosAngeles SanDiego 100
SanDiego SanFrancisco 100
LosAngeles SanFrancisco 200

Imagine, for example, that the edge cost represents the price of a one-way flight from the departure city to the arrival city (for a fairly unusual airline). Then there are two paths from NYC to San Francisco with minimum cost of 600:

NYC to LosAngeles to SanDiego to SanFrancisco
NYC to LosAngeles to SanFrancisco
Your program should return the second option, since that journey has only 2 legs rather than 3 legs.

Your assignment is to implement an efficient version of Dijkstra's algorithm with extra book-keeping to allow the minimum distance path with fewest edges to be found. Only a few modifications to the pseudocode given in the textbook should be required. The main fact to keep in mind is the following (which is what makes the strategy in Dijkstra's algorithm work):

If \( s = v_1 \ v_2 \ldots \ v_i \ldots v_N = v \) is a minimum length path from \( s \) to \( v \), then for every vertex \( v_i \) along the path, it must also be true that \( v_1 \ v_2 \ldots \ v_i \) is a minimum length path from \( s \) to \( v_i \).

Suggestion: first read the implementation of the Dijkstra's algorithm as given in the text, then think about what changes must be made to guarantee that you will also find the minimum cost path with fewest edges. Note that you should not keep track of all minimum length paths to do this.