1. (10 points) You run the strongly connected components algorithm on the following graph. Assume that all ties are broken alphabetically (in DFS, for instance). Show the components you get, in the order you get them, along with all relevant work. Show the final underlying component graph. The graph is given in adjacency list format.

   GRAPH ADJACENCY LIST HERE

2. (10 points) Simulate the DAG Single Source Shortest Path algorithm on the graph below, using vertex s as a source. Use Kahn’s topological sort algorithm. Assume that initially, vertices are traversed alphabetically, and that within a vertex, its edge list is also traversed alphabetically. The graph is given as a list of outgoing edges for each vertex. Show all work, including the Queue, and all intermediate distances the algorithm temporarily stores. The graph is given in adjacency matrix format, with weights. (Blank means no edge.) Show work.

   GRAPH ADJACENCY MATRIX HERE

3. Consider the following graph for each of the two subproblems. Each subproblem is independent of the other. The graph is drawn twice, so that you can use one for work for each subproblem.

   [a] (10 points) You are running Prim’s algorithm on the graph below and left, starting with vertex (some vertex named here). You are about to take vertex (some other vertex named here) out of the min-heap, but have not done so yet. Show all work done so far, and for that instant in time, categorize each edge (every one of them) of the graph from the following list:

   (a) Already known to be in the graph.
   (b) The edge has not been seen yet.
   (c) The edge has been seen, and is known to not be in the graph.
   (d) The edge has been seen, and is stored with a vertex in the heap.

   GIVE GRAPHS HERE

   [b] (10 points) You are in the middle of running Kruskal’s Minimum Weight Spanning Tree algorithm, using the Disjoint Set data structure, with union by size and path compression. The graph you are running on is above. You are about to consider the edge with weight X. Draw the corresponding disjoint set data structure for the graph, before and after the edge with weight X. Assume that, for each edge of the graph, the edge is called with the alphabetically first vertex first, and that for sets of the same size, the first set joins to the second.

4. (10 points) You want to code Dijkstra’s Single Source Shortest Path algorithm, using a min-heap to store path lengths. Unfortunately, you are running the algorithm on a graph that contains negative edge weights. Your code never checks for negative edge weights, and if you relax an edge leading to a vertex that isn’t in the heap anymore? Your code changes the vertex’s distance estimate, but doesn’t signal an error, nor put the vertex back into the heap. Show what happens when running the algorithm on the graph below. Show all work.

   GIVE GRAPH HERE
5. (10 points) You are in the middle of running the Floyd-Warshal algorithm. You have just completed calculating $D^3$, $\Pi^3$, and $\Phi^3$. The weather outside is just lovely. Follow the algorithm to calculate values for $D^4$, $\Pi^4$, and $\Phi^4$. Fill in those values below. (You only need to do it once, I just give the table twice to help you make it legible if it comes out poorly once. Your answer should be legible.)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>5</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>-</td>
<td>4</td>
<td>3</td>
<td>0</td>
<td>4</td>
<td>5</td>
<td>-</td>
<td>3</td>
<td>4</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

6. (10 points) For the following graph, you are in the midst of running the Bellman Ford algorithm, from source vertex $s$. Taking vertices (including the source) in alphabetical order, relax all outgoing edges once. What distance and $\pi$ values are stored with each vertex? Show all work, including intermediate distance estimates.

GIVE GRAPH HERE

7. (10 points) You are running Breadth First Search on the following graph. When considering a vertex, instead of adding all of its adjacent, undiscovered vertices to a queue (FIFO), add them to a stack (LIFO) instead. Show the stack at each step of the way, and show the altered discovery tree that it finds.

GIVE GRAPH HERE