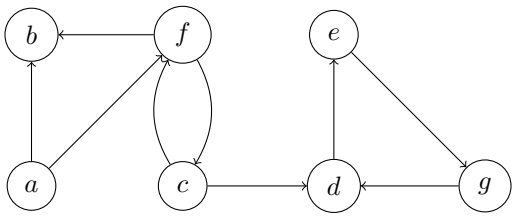
# DSC 40B - Discussion 07

**Problem 1.**

Consider a *breadth*-first search on the graph shown in the figure, starting with node $c$.



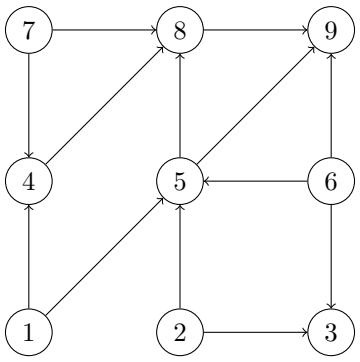

**a)** Suppose you call bfs_shortest_paths(graph, 'c') on the graph above. This function returns dictionaries distance and predecessor. Write down the contents of these dictionaries as they are when the function exits.

```
def bfs_shortest_paths(graph, source):
    status = {node:'undiscovered' for node in graph.nodes}
    distance= {node:float('inf') for node in graph.nodes}
    predecessor = {node: None for node in graph.nodes}
    status[source] = 'pending'
    distance[source]=0
    pending = deque([source])
    # while there are still pending nodes
    while pending:
        u = pending.popleft()
        for v in graph.neighbors(u):
            # explore edge (u,v)
            if status[v] == 'undiscovered':
                status[v]='pending'
                distance[v]=distance[u]+1
                predecessor[v]=u
                # append to right
                pending.append(v)
        status[u]='visited'
    return predecessor, distance
```

**b)** Mark the BFS trees produced on executing BFS on this graph.

**Problem 2.**

Consider the following directed graph.




**a)** Run Full_DFS on the graph above. Make a bold arrow from node $u$ to node $v$ if $u$ is the predecessor of node $v$ in DFS. Use the convention that a node's neighbors are processed in ascending order by label.

**b)** Fill in the table below so that it contains the start and finish times of each node after a Full_DFS is performed on the above graph. Assume node 1 as the source for the first DFS call. Begin your start times with 1.

| Node | Start | Finish |
|---|---|---|
| 1 | | |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |

| Node | Start | Finish |
|---|---|---|
| 6 | | |
| 7 | | |
| 8 | | |
| 9 | | |

**Problem 3.**

Given an undirected graph G=(V,E), give an algorithm to find if the graph is disconnected.