Problem 1

Run DFS on the graph above starting from $q$. Mark the start and finish times of each vertex and classify each edge as a forward, back, tree or cross edge. You may assume we process vertices in alphabetical order.
Problem 2
Design an algorithm to find the shortest path between nodes $u$ and $v$ in a connected, unweighted graph.

Solution
Since the graph is unweighted, we can just run BFS starting from $u$ and for each node $x$ that we visit, we just keep a pointer to its parent node (the node we visited $x$ from). When we reach $v$, we stop and find the shortest path by backtracking through the pointers that we kept (i.e. we could see that $v$'s parent was $d$, $d$'s parent was $c$, and $c$'s parent was $u$, so our path would be $u \rightarrow c \rightarrow d \rightarrow v$). We just do a BFS and backtrack no more than $O(n)$ times (the longest path in a graph is $n - 1$ edges), so this algorithm also runs in linear time.

Problem 3: Finding a Cycle in a Directed Graph
Design an algorithm to determine if a directed graph $G$ has a cycle, and return a cycle if one exists.

Solution
We can use DFS for this question. We will run DFS from an arbitrary vertex $s$, while also maintaining a parent pointer array $A$. During our DFS traversal, if we go from a node $u$ to a neighbor $v$, we will set $A[v] = u$. If we ever see a back edge $(u, v)$, we know we have found a cycle. We can then return the cycle by creating a list $L$ and following the parent pointers until we get back to $v$, at each step adding the current node to the front of the list. Once we reach vertex $v$, then $L$ must contain every node in the cycle so we return it. If we never see a back edge during our DFS traversal, then there are no cycles in $G$ so we return nothing.