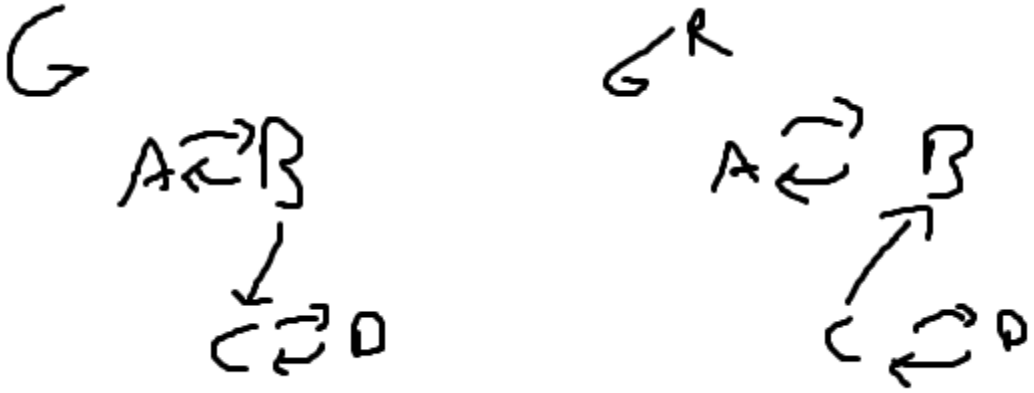
1. False. Consider the graphs below. The post-order of G is DCBA. The reverse-post-order of G^R is BADC. If this conjecture were true, Kosaraju's algorithm would have been a lot simpler (no need to form the reverse graph at all).



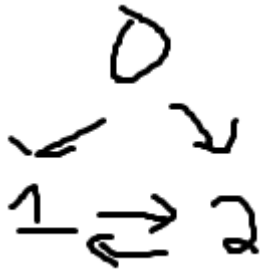
2. Suppose we run DFS from some starting vertex v and let w be the first vertex that finishes. This vertex is obviously a dead end, and thus must either be a leaf, or it must only point back at some previously visited node. Thus, we can remove this vertex and the rest of the graph will remain connected.

3. We first create a subroutine that finds the shortest cycle starting and ending with s. This is relatively straightforward, we simply run BFS from vertex s, with a little extra logic to check for any edges that point back to s. If you encounter any edge back to s, you've found a cycle starting and ending at s. Record the length and terminating vertex of the cycle if it is shorter than the previous best. Repeat for each vertex.

4. The simplest way is to create two copies of the graph. We think of one of these graphs as the "odd" graph and one as the "even" graph. For each edge $v \rightarrow w$ in the original graph, we create an edge from $v_{odd} \rightarrow w_{even}$ and $v_{even} \rightarrow w_{odd}$. The overall effect is that if a vertex v_{odd} is marked at the end of a DFS, then that vertex can be reached via an odd length path.

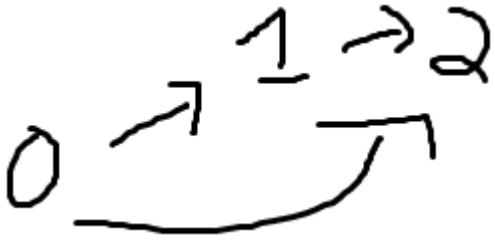
   Another approach (common on the Fall 2014 final) is as follows: Run BFS, but maintain an evenQueue and an oddQueue and give each vertex an even and odd marker. We start by marking the source as even and enqueuing it on the evenQueue. From there, we dequeue each vertex v from the evenQueue (only the source on the first iteration); then for each neighbor w of v, we then mark each w as odd and enqueue it on the oddQueue, unless w has already been marked as odd (in which case we ignore it). We next do the same thing with the oddQueue, dequeuing each vertex v, and marking and enqueuing each neighbor w as long as it is not even. We alternate between the two queues until both are empty.

5. It is something else. Consider the graph



DFS would need to return either: 0->1->2 and 0->1, or 0->2->1 and 0->2. The algorithm shown returns the paths 0->1 and 0->2. Thus it is not DFS.

Now consider the graph below:



BFS would need to return 0->2 for the path to 2, but this algorithm might return 0->1->2 depending on the adjacency lists. Thus it is not BFS (though it is quite close in behavior).

6. Still not sure if we should include this problem. It gets a bit theoretical. The basic argument is covered pretty well here:
http://www.bowdoin.edu/~ltoma/teaching/cs231/fall08/Homeworks/H9-sol%203.pdf

We could just replace this problem with problem 12 of the Spring 13 midterm, which is to find the number of shortest paths on an undirected graph:
http://www.cs.princeton.edu/courses/archive/spring13/cos226/exams/fin-s13.pdf