Module 2 - Graphs: Basic Written HW

1. Let G be an undirected graph with n nodes (let's assume n is even). Prove or provide a counterexample for the following claim: If every node of G has a degree of at least $\frac{n}{2}$, then G must be connected.

2. Most graph algorithms that take an adjacency-matrix representation as input require time $\Omega(V^2)$, but there are some exceptions. Show how to determine whether a directed graph G contains a universal sink—a vertex with in-degree $|V| - 1$ and out-degree 0 in time $O(V)$, given an adjacency matrix for G.

3. The textbook describes two variables that can be associated with each node in a graph G during the execution of *Depth-First Search*: discovery time ($v.d$) and finish time ($v.f$). These are integer values that are unique. Every time a node is discovered (i.e., DFS sees the node for the first time) that node's $v.d$ is set to the next available integer. When DFS is finished exploring ALL of this node's children, $v.f$ is set to the next available integer.

   For this question, consider a single edge in a graph G after DFS finishes executing. You might need to reference the textbook or slides for definitions of tree edge, forward edge, back edge, and cross edge. Argue that each edge $e = (u, v)$ is:

   1. A tree edge or forward edge if and only if $u.d < v.d < v.f < u.f$
   2. A back edge if and only if $v.d \leq u.d < u.f \leq v.f$
   3. A cross edge if and only if $v.d < v.f < u.d < u.f$

   You can describe your answers intuitively, but your answers must be clearly articulated.

4. For a given undirected graph G, prove that the depth of a DFS tree cannot be smaller than the depth of the BFS tree. (Clearly state your proof strategy or technique.)

   Tom's comments: Pretty easy to argue. (If BFS tree smaller, then at least one vertex v must be at a lower level in the BFS tree, but we know BFS shows shortest paths.) Maybe better as a quiz question if we want a very simple proof on the first version of Quiz 2.

5. Given an undirected graph G, the *eccentricity* of a node v is the largest of the shortest possible distances from v to any other node in the graph. The minimum eccentricity in the graph is called the *graph radius* for G. All the nodes in G that have eccentricity equal to the graph radius form a set called the *graph center* of G.

   Describe (using pseudo-code or a very clea rtext explanation) an efficient algorithm to find the graph center of a graph G and describe its complexity. (Note: you must make use of algorithms studied in this module, and not re-invent the wheel.)

   Tom's comments: We could rename these graph terms if we wanted to hinder web searches for the algorithm. Solution involves calling BFS on each node, and remember the longest path in BFS tree as the eccentricity for the start-node. The find the minimum of all node's eccentricity, then find all nodes that have this minimum. $\Theta(V \times E)$

6. A graph G's *girth* is the number of edges in the shortest cycle in G. Describe an efficient algorithm to find the girth of a graph in $\Theta(V \times E)$. (Note: you must make use of algorithms studied in this module, and not re-invent the wheel.)

   Tom's comments: They need to realize that when they hit a leaf in the BFS tree, they can check to see if it connects to the start node. They also need to know they need to run BFS starting at each vertex.
   We could change or not use "girth" if we wanted to hinder web searches for the algorithm. This is from the Johnsonbaugh textbook, which is not so popular, but I see Chegg when searching for solutions. What can you do... BTW, is type-setting the multiplication as $\Theta(V \times E)$ better than $\Theta(VE)$?

7. In the CLRS code for *DFS-Visit* at line 8, we have completed recursively visiting all then un-visited vertices adjacent to vertex u, so we change u's color to black. For this problem, consider what happens if instead we change the color of u to *white* at this line.

   Explore what this causes to happen by tracing this altered algorithm on a small connected undirected graph. After you have done that, submit answers to the following questions.

   A. In no more than a few sentences, clearly describe what the "DFS" tree from an initial node s found by this algorithm represents. (Note: we put DFS in quotes here because with this change this algorithm is no longer DFS!)

   B. Give a mathematical formula (not just an order class) for the number of leaves in this tree for a worst-case input graph. Describe this worst-case input graph. (Again, tracing this algorithm on small connected graphs may help you find the answer more quickly.)

   Tom's comments: For (a) we're looking for an answer along these lines: All possible simple paths in the graph from s. For (b) the answer should be: for a complete graph, $(n-1)!$