Problem 1: Given an undirected graph \( G = (V, E) \) you want to color the nodes Green and Yellow in such a way that two nodes connected by an edge must have a different color. Use BFS to show how to determine if such a coloring is possible, and find it if it exists.

Solution: Pick a node \( v \) arbitrarily and color it arbitrarily (say G). Then using BFS, find all the nodes at distance 1 from \( v \) and color them Y. Continue this way alternating each color. When you are done, check if there is a "bad" edge (i.e. an edge connecting two nodes with the same color). If there is such a bad edge then it is not possible to color the graph with 2 colors: the proof is simple, since all our choices during the BFS "pass" were forced. If there is no bad edge then we just found a good coloring.

Problem 2: Given a directed acyclic graph \( G = (V, E) \) you want to find the length \( L \) of the longest path in the graph.

- Use DFS to get a topological sort of the nodes \( v_1, \ldots, v_n \). Denote with \( L[i] \) the length of the longest path starting at \( v_i \). Express \( L(i) \) as a function of \( L[j] \) for \( j > i \).
- Use dynamic programming to fill the array \( L[1], \ldots, L[n] \) and use these values to compute \( L \).
- What is the running time of your algorithm as a function of \( n = |V| \) and \( m = |E| \)?

Solution:

- Look at all the edges leaving \( v_i \). Since the nodes are sorted topologically, these edges will go to nodes \( v_j \) for \( j > i \). Therefore the longest path starting at \( i \) will be a path composed by one such edge going to say \( v_j \), followed by the longest path starting at \( v_j \). Therefore
  \[
  L[i] = 1 + \max_{j: v_j \in \text{Adj}[v_i]} [L[j]]
  \]
- The above equation gives an immediate dynamic programming algorithm. Obviously \( L[n] = 0 \), and then you can fill the array \( L \) by going backwards. Once we filled the entire array, obviously \( L = \max_i [L[i]] \).
- It takes \( O(n + m) \) to perform the topological sort. It takes \( O(n + m) \) to fill the array \( L \) (notice that to fill position \( i \) we only look at the edges coming out of \( v_i \) so each steps costs \( O(m_i) \) where \( m_i \) is the number of edges coming out of \( v_i \), since \( m = \sum_i m_i \) we have the claimed complexity). Computing \( L \) takes time \( O(n) \). Therefore the running time is \( O(n + m) \).

Problem 3: Prove or disprove the following fact: Given a graph \( G \), the tree generated by a DFS algorithm always has height larger or equal than the height of the tree generated by a BFS algorithm.

Solution: In general this is false. Consider a "star" graph (i.e. a graph with nodes \( u_0, u_1, \ldots, u_n \), with edges \( (u_0, u_i) \) for all \( i \). In this case if you start DFS from \( u_0 \), the tree has height 1 (with \( u_0 \) being the root, and all the other nodes being leaves). If you start BFS from say \( u_1 \) the tree will have height 2, with \( u_1 \) as the root, \( u_0 \) as its only child, and all the other nodes as leaves (children of \( u_0 \)).

For any graph if you start BFS and DFS on the same node, then it is true that the tree generated by DFS always has height larger or equal than the height of the tree generated by a BFS algorithm.