1. Suppose that you start with an arbitrary forest of “up trees,” containing \( n \) nodes altogether. These trees are not necessarily balanced at all. You then perform \( m \) find operations, using path compression. Show that executing these find operations takes time \( O(n + m) \).

2. There are \( n \) people that we want to partition into two groups. We also have a list of \( m \) pairs of people who are incompatible, and so cannot be assigned to the same group. Design and analyze a linear-time algorithm (i.e., \( O(m + n) \) time) to solve this problem: your algorithm should determine if such a partition is possible, and if so, output the partition. Hint: use BFS as a subroutine.

3. Let \( G = (V, E) \) be a connected, undirected graph. Give an \( O(|V| + |E|) \)-time algorithm to compute a path in \( G \) that traverses each edge in \( E \) exactly once in each direction. Hint: use BFS as a subroutine.

4. Consider the following directed graph:

Show how depth-first search works on this graph. Assume that the main loop of the DFS procedure processes the vertices in alphabetical order, and assume that each adjacency list is ordered alphabetically. Show the discovery and finishing times for each vertex. Also, redraw the graph to show the structure of the DFS forest, as was done in the example in the class notes: tree edges should be solid, while all other edges should be dashed. You should also indicate which edges are forward, back, and cross edges.

5. Another way to perform topological sorting on a directed acyclic graph \( G = (V, E) \) is to repeatedly find a vertex of in-degree 0, output it, and remove it and all of its outgoing edges from the graph. Explain how to implement this idea so that it runs in time \( O(|V| + |E|) \). What happens to this algorithm if \( G \) has cycles?

6. **Sightseeing.** You are a tourist in a city, and you want to take in as many sights as possible on a single tour. Let’s model this as a directed graph \( G = (V, E) \), where the nodes in the graph represent the sights, and the edges represent roads between them (the graph is directed, because some roads may be one-way). For a path \( p \) in the graph, define the **sightseeing value** of \( p \) to be the number of **distinct** nodes in \( p \). For a node \( v \in V \), define the **sightseeing value** of \( v \) to be the maximum sightseeing value of any path starting at \( v \). Design and analyze an algorithm to find a node of maximum sightseeing value. Your algorithm should run in time \( O(|V| + |E|) \).

   Hint: use a strongly connected components algorithm.

7. **Honor’s exercise:** Problem 22-1(b) on p. 621 of CLRS.