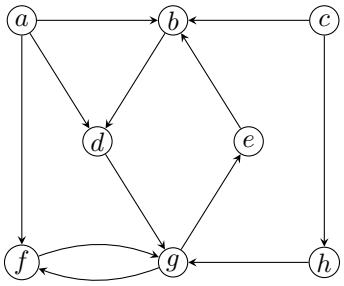
Problem 1 (20 points). You are given the following directed graph $G = (V, E)$.

1. Use adjacency list to represent G.
2. Run the DFS (with-time) algorithm using your above adjacency list as input. Give the pre/post interval for each vertex in V. Classify each edge in E as a tree edge, forward edge, back edge, or cross edge.



Problem 2 (20 points). For a given graph G, we know that different adjacency list representation of G leads to different DFS runs, and consequently may result in different search forests. For each of the following questions, either give such an instance, or prove that there does not exist such instance.

1. Can you design an instance of a directed graph, such that one DFS run reveals forward-edge(s), but another DFS run does not?
2. Can you design an instance of a directed graph, such that one DFS run reveals cross-edge(s), but another DFS run does not?
3. Can you design an instance of a directed graph, such that one DFS run reveals back-edge(s), but another DFS run does not?
4. Can you design an instance of a directed graph, such that two DFS runs reveals different number of tree-edges?

Problem 3 (20 points).

1. Given an undirected graph $G = (V, E)$ and an edge $e = (u, v) \in E$, design a linear time algorithm to determine whether there exists a cycle in G that contains e.
2. Given a directed graph $G = (V, E)$ and an edge $e = (u, v) \in E$, design a linear time algorithm to determine whether there exists a cycle in G that contains e.

Problem 4 (20 points). In an undirected graph, the *degree* of vertex v, denoted as $d(v)$, is defined as the number of neighbors v has, or equivalently, the number of edges adjacent to v.

1. Given an undirected graph $G = (V, E)$ represented as an adjacent list, design an $O(|E|)$ time algorithm to compute the degree for all vertices.
2. For vertex $v \in V$, define $d^2(v)$ as the sum of the degree of all neighbors of v, i.e., $d^2(v) = \sum_{u:(v,u)\in E} d(u)$. Design an $O(|E|)$ time algorithm to compute $d^2(\cdot)$ for all vertices.

Problem 5 (20 points). We have three containers with sizes of 10 pints, 7 pints, and 4 pints, respectively. The 7-pint and 4-pint containers start with full of water, but the 10-pint container is initially empty. We are allowed the following operation: pouring the contents of one container into another, stopping only when the source container is empty, or the destination container is full. We want to determine, is there a sequence of such operations that leaves exactly 2 pints in the 7-pint or 4-pint container.

1. Model this task as a graph problem: give a precise definition of the graph (i.e., what are the vertices and what are the edges), and then state the question about this graph that needs to be answered.

2. What algorithm should be applied to solve the task?