1. The adjacency list representation of a graph $G$, which has 7 vertices and 10 edges, is:

$$
\begin{align*}
a &: \to d, e, b, g \\
b &: \to e, c, a \\
c &: \to f, e, b, d \\
d &: \to c, a, f \\
e &: \to a, c, b \\
f &: \to d, c \\
g &: \to a
\end{align*}
$$

(a) Show the tree produced by depth-first search when it is run on the graph $G$, using vertex $a$ as the source. You must use the adjacency list representation given above. (Recall that the DFS tree can depend on the order of vertices in the adjacency lists; for this problem you are required to use the adjacency lists as given above.) Note that in this case you are running DFS on an undirected graph and not a directed one so you will have to slightly modify the algorithm you learnt in class.

(b) In the DFS tree of item (a), show the edges of the graph $G$ which are not present in the DFS tree by dashed lines.

2. Show that depth-first search of an undirected graph $G$ can be used to identify the connected components of $G$. More precisely, show how to modify depth-first search so that each vertex $v$ is assigned an integer label $cc[v]$ between 1 and $k$, where $k$ is the number of connected components of $G$, such that $cc[u] = cc[v]$ if and only if $u$ and $v$ are in the same connected component.

3. Prove that if $G$ is a connected undirected graph, then each of its edges is either in the depth-first search tree or is a back edge.
4. Give a simple example of a directed graph with negative-weight edges for which Dijkstra’s algorithm produces incorrect answers. Why does the correctness proof of Dijkstra’s algorithm not go through when negative-weight edges are allowed?

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 (why does such a vertex always exist?) 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?