Topological Sorting

directed graph, or ***digraph*** for short, is a graph with directions specified for all its edges (Figure 4.5a is an example). The adjacency matrix and adjacency lists are still two principal means of representing a digraph. There are only two notable differences between undirected and directed graphs in representing them: (1) the adjacency matrix of a directed graph does not have to be symmetric; (2) an edge in a directed graph has just one (not two) corresponding nodes in the digraph's adjacency lists. Depth-first search and breadth-first search are principal traversal algorithms for traversing digraphs as well, but the structure of corresponding forests can be more complex than for undirected graphs.
Note that a back edge in a DFS forest of a directed graph can connect a vertex to its parent. Whether or not it is the case, the presence of a back edge indicates that the digraph has a directed cycle. A ***directed cycle*** in a digraph is a sequence of three or more of its vertices that starts and ends with the same vertex and in which every vertex is connected to its immediate predecessor by an edge directed from the predecessor to the successor. For example, *a, b, a* is a directed cycle in the digraph in Figure 4.5a. Conversely, if a DFS forest of a digraph has no back
edges, the digraph is a ***dag***, an acronym for ***directed acyclic graph***. For topological sorting to be possible, a digraph in question must be a dag. It turns out that being a dag is not only necessary
but also sufficient for topological sorting to be possible; i.e., if a digraph has no directed cycles, the topological sorting problem for it has a solution

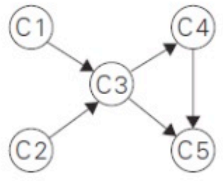

FIGURE 4.6 Digraph representing the prerequisite structure of five courses.

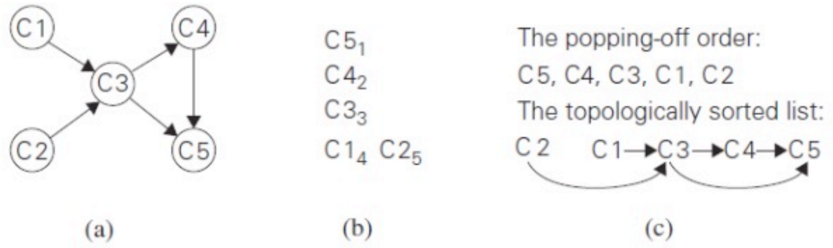

FIGURE 4.7 (a) Digraph for which the topological sorting problem needs to be solved. (b) DFS traversal stack with the subscript numbers indicating the popping-off order. (c) Solution to the problem.

There are two efficient algorithms that both verify whether a digraph is a dag and, if it is, produce an ordering of vertices that solves the topological sorting problem. The first algorithm is a simple application of depth-first search: perform a DFS traversal and note the order in which vertices become dead-ends (i.e., popped off the traversal stack). Reversing this order yields a solution to the topological sorting problem, provided, of course, no back edge has been encountered during the traversal. If a back edge has been encountered, the digraph is not a dag, and topological sorting of its vertices is impossible.
The second algorithm is based on a direct implementation of the decrease-(by

one)-and-conquer technique: repeatedly, identify in a remaining digraph a ***source***, which is a vertex with no incoming edges, and delete it along with all the edges outgoing from it. (If there are several sources, break the tie arbitrarily. If there are none, stop because the problem cannot be solved The order in which the vertices are deleted yields a solution to the topological sorting problem. The application of this algorithm to the same digraph representing the five courses is given in Figure 4.8. Note that the solution obtained by the source-removal algorithm is different from the one obtained by the DFS-based algorithm. Both of them are correct, of course; the topological sorting problem may have several alternative solutions.