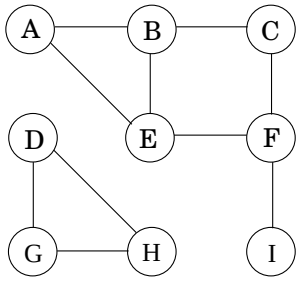
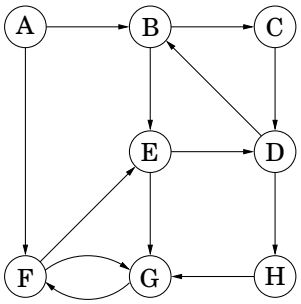
Exercises

3.1. Perform a depth-first search on the following graph; whenever there's a choice of vertices, pick the one that is alphabetically first. Classify each edge as a tree edge or back edge, and give the `pre` and `post` number of each vertex.

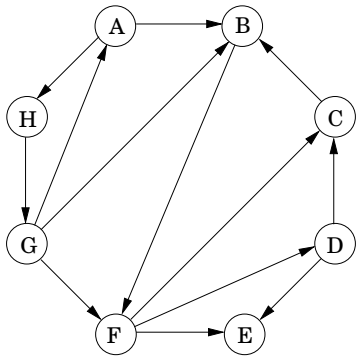


3.2. Perform depth-first search on each of the following graphs; whenever there's a choice of vertices, pick the one that is alphabetically first. Classify each edge as a tree edge, forward edge, back edge, or cross edge, and give the `pre` and `post` number of each vertex.

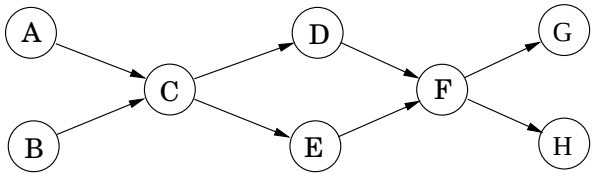
(a)



(b)



3.3. Run the DFS-based topological ordering algorithm on the following graph. Whenever you have a choice of vertices to explore, always pick the one that is alphabetically first.



(a) Indicate the `pre` and `post` numbers of the nodes.

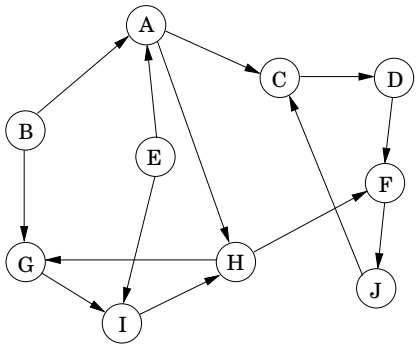
(b) What are the sources and sinks of the graph?

(c) What topological ordering is found by the algorithm?

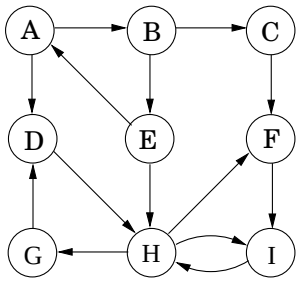
(d) How many topological orderings does this graph have?

3.4. Run the strongly connected components algorithm on the following directed graphs G. When doing DFS on G^R: whenever there is a choice of vertices to explore, always pick the one that is alphabetically first.

(i)



(ii)



In each case answer the following questions.

(a) In what order are the strongly connected components (SCCs) found?

(b) Which are source SCCs and which are sink SCCs?

(c) Draw the "metagraph" (each meta-node is an SCC of G).

(d) What is the minimum number of edges you must add to this graph to make it strongly connected?

3.5. The *reverse* of a directed graph $G = (V, E)$ is another directed graph $G^R = (V, E^R)$ on the same vertex set, but with all edges reversed; that is, $E^R = \{(v, u) : (u, v) \in E\}$.

Give a linear-time algorithm for computing the reverse of a graph in adjacency list format.

3.6. In an undirected graph, the *degree* $d(u)$ of a vertex u is the number of neighbors u has, or equivalently, the number of edges incident upon it. In a directed graph, we distinguish between the *indegree* $d_{in}(u)$, which is the number of edges into u, and the *outdegree* $d_{out}(u)$, the number of edges leaving u.

(a) Show that in an undirected graph, $\sum_{u \in V} d(u) = 2|E|$.

(b) Use part (a) to show that in an undirected graph, there must be an even number of vertices whose degree is odd.

(c) Does a similar statement hold for the number of vertices with odd indegree in a directed graph?

3.7. A *bipartite graph* is a graph $G = (V, E)$ whose vertices can be partitioned into two sets ($V = V_1 \cup V_2$ and $V_1 \cap V_2 = \emptyset$) such that there are no edges between vertices in the same set (for instance, if $u, v \in V_1$, then there is no edge between u and v).

(a) Give a linear-time algorithm to determine whether an undirected graph is bipartite.

(b) There are many other ways to formulate this property. For instance, an undirected graph is bipartite if and only if it can be colored with just two colors.

Prove the following formulation: an undirected graph is bipartite if and only if it contains no cycles of odd length.

(c) At most how many colors are needed to color in an undirected graph with exactly *one* odd-length cycle?

3.8. *Pouring water.* We have three containers whose sizes are 10 pints, 7 pints, and 4 pints, respectively. The 7-pint and 4-pint containers start out full of water, but the 10-pint container is initially empty. We are allowed one type of 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 know if there is a sequence of pourings that leaves exactly 2 pints in the 7- or 4-pint container.

(a) Model this as a graph problem: give a precise definition of the graph involved and state the specific question about this graph that needs to be answered.

(b) What algorithm should be applied to solve the problem?

(c) Find the answer by applying the algorithm.

3.9. For each node u in an undirected graph, let `twodegree`$[u]$ be the sum of the degrees of u's neighbors. Show how to compute the entire array of `twodegree`$[\cdot]$ values in linear time, given a graph in adjacency list format.

3.10. Rewrite the `explore` procedure (Figure 3.3) so that it is non-recursive (that is, explicitly use a stack). The calls to `previsit` and `postvisit` should be positioned so that they have the same effect as in the recursive procedure.

3.11. Design a linear-time algorithm which, given an undirected graph G and a particular edge e in it, determines whether G has a cycle containing e.

3.12. Either prove or give a counterexample: if $\{u, v\}$ is an edge in an undirected graph, and during depth-first search `post`$(u) <$`post`(v), then v is an ancestor of u in the DFS tree.

3.13. *Undirected vs. directed connectivity.*

(a) Prove that in any connected undirected graph $G = (V, E)$ there is a vertex $v \in V$ whose removal leaves G connected. (*Hint:* Consider the DFS search tree for G.)

(b) Give an example of a strongly connected directed graph $G = (V, E)$ such that, for every $v \in V$, removing v from G leaves a directed graph that is not strongly connected.

(c) In an undirected graph with 2 connected components it is always possible to make the graph connected by adding only one edge. Give an example of a directed graph with two strongly connected components such that no addition of one edge can make the graph strongly connected.

3.14. The chapter suggests an alternative algorithm for linearization (topological sorting), which repeatedly removes source nodes from the graph (page 101). Show that this algorithm can be implemented in linear time.

3.15. The police department in the city of Computopia has made all streets one-way. The mayor contends that there is still a way to drive legally from any intersection in the city to any other intersection, but the opposition is not convinced. A computer program is needed to determine whether the mayor is right. However, the city elections are coming up soon, and there is just enough time to run a *linear-time* algorithm.

(a) Formulate this problem graph-theoretically, and explain why it can indeed be solved in linear time.

3.22. Give an efficient algorithm which takes as input a directed graph $G = (V, E)$, and determines whether or not there is a vertex $s \in V$ from which all other vertices are reachable.

3.23. Give an efficient algorithm that takes as input a directed acyclic graph $G = (V, E)$, and two vertices $s, t \in V$, and outputs the number of different directed paths from s to t in G.

3.24. Give a linear-time algorithm for the following task.

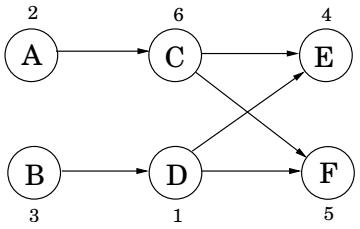
Input: A directed acyclic graph G

Question: Does G contain a directed path that touches every vertex exactly once?

3.25. You are given a directed graph in which each node $u \in V$ has an associated *price* p_u which is a positive integer. Define the array `cost` as follows: for each $u \in V$,

$$\texttt{cost}[u] = \text{price of the cheapest node reachable from } u \text{ (including } u \text{ itself).}$$

For instance, in the graph below (with prices shown for each vertex), the `cost` values of the nodes A, B, C, D, E, F are $2, 1, 4, 1, 4, 5$, respectively.



Your goal is to design an algorithm that fills in the *entire* `cost` array (i.e., for all vertices).

(a) Give a linear-time algorithm that works for directed *acyclic* graphs. (*Hint:* Handle the vertices in a particular *order*.)

(b) Extend this to a linear-time algorithm that works for all directed graphs. (*Hint:* Recall the "two-tiered" structure of directed graphs.)

3.26. An *Eulerian tour* in an undirected graph is a cycle that is allowed to pass through each vertex multiple times, but must use each edge exactly once.

This simple concept was used by Euler in 1736 to solve the famous Konigsberg bridge problem, which launched the field of graph theory. The city of Konigsberg (now called Kaliningrad, in western Russia) is the meeting point of two rivers with a small island in the middle. There are seven bridges across the rivers, and a popular recreational question of the time was to determine whether it is possible to perform a tour in which each bridge is crossed *exactly once*.

Euler formulated the relevant information as a graph with four nodes (denoting land masses) and seven edges (denoting bridges), as shown here.