Practice Problems 2: Solutions

Problem 1. In this problem you will simulate the strong-components algorithm given in class on the digraph shown in Fig. 1.

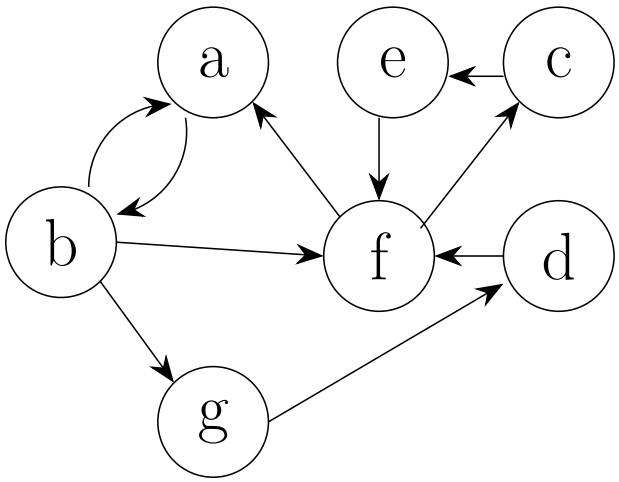


Figure 1: A directed graph

(a) Show the result of applying DFS to G. Label each vertex u with its discovery and finish times ($d[u]/f[u]$). You need only show the *final* DFS tree, not the intermediate results. Whenever you have a choice of which vertex to visit next, select the lowest vertex in alphabetic order.

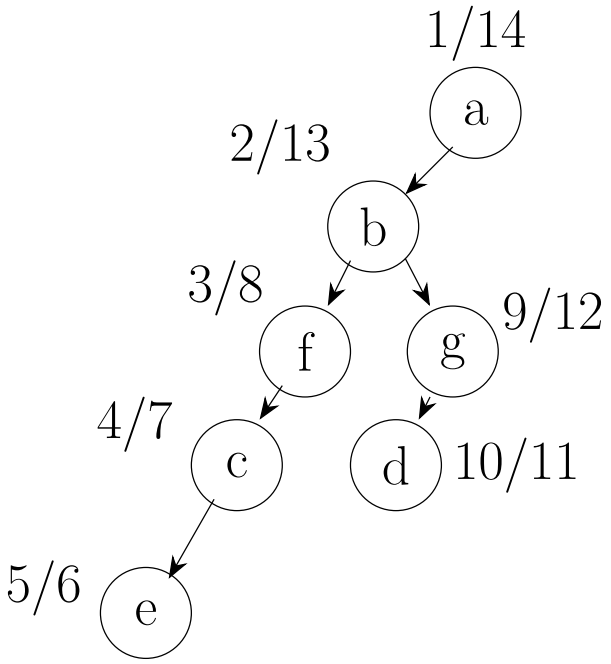
(b) Draw the reverse graph G^R. (Be careful to note the directions of the edges.)

(c) Show the result of running DFS on G^R, where the main program selects the next vertex to visit based on decreasing order of finish times from part (a).

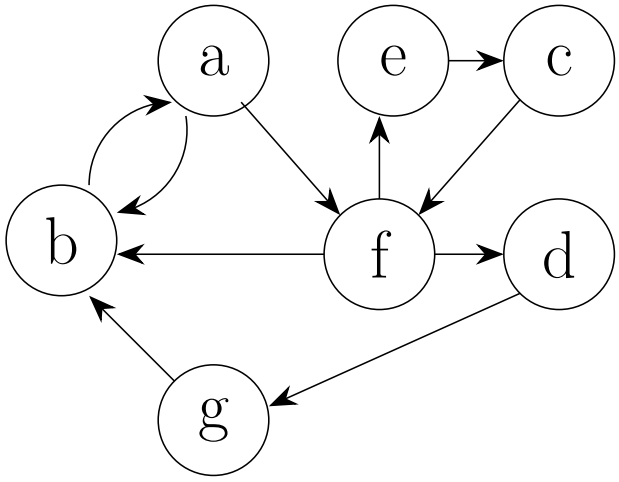
(d) Illustrate (e.g., by circling them or listing them out) the strong components of G.

Solution:

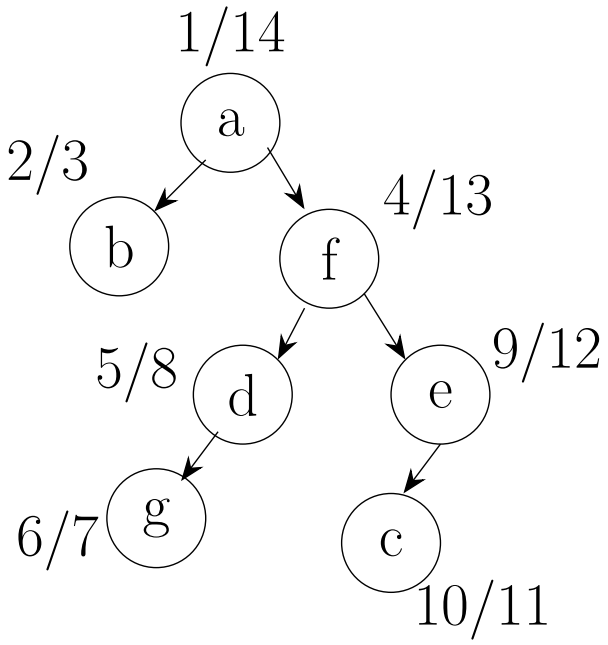
(a) The DFS tree of G, with each vertex labeled $d[u]/f[u]$. The time counter starts at 0 and is incremented both when a vertex is discovered and when it is finished.



(b) The reverse graph G^R:



(c) In decreasing order of finish time from part (a), the main program considers the vertices in the order a, b, g, d, f, c, e. The DFS on G^R started at a already reaches every vertex, so the result is a single DFS tree:



(d) The entire graph is a single strong component, $\{a, b, c, d, e, f, g\}$. This is evident from the fact that the DFS from a reaches every vertex both in G and in G^R: every vertex is reachable from a and can reach a, so all vertices are mutually reachable.

Problem 2. A **source** of a directed graph is a node with no incoming edges. A **sink** of a directed graph is a node with no outgoing edges.

(a) Prove that if $G = (V, E)$ is a directed acyclic graph, G contains at least one source and at least one sink.

(b) Further prove that if $G = (V, E)$ is a directed acyclic graph, then for every vertex v there exists a source s and a sink t such that there exists a path from s to v and a path from v to t.

Solution:

(a) Suppose for contradiction that G is a DAG with no sink, so every vertex has at least one outgoing edge. Since G is acyclic, no path can repeat a vertex, so every path has at most $|V|$ vertices and a longest path P exists. Let v be the last vertex of P. As G has no sink, v has an outgoing edge (v, v'), and v' cannot already lie on P (otherwise G would contain a cycle). Appending v' to P gives a longer path, contradicting the maximality of P. Thus every DAG contains a sink. The argument may be mirrored (extending a longest path backwards from its first vertex) to show the existence of a source.

(b) Consider any DAG G and vertex v. Because G is acyclic, there is a longest path starting from v; let it end at some vertex v'. If v' had any outgoing edge, the path could be extended (the new endpoint cannot already lie on the path, since G is acyclic), contradicting its maximality. Thus v' is a sink reachable from v. The argument may be mirrored, using a longest path ending at v, to show that some source has a path to v.

Problem 3. For both parts below, assume that graphs and digraphs are represented using an adjacency list. Given a graph or digraph $G = (V, E)$, let $n = |V|$ and $m = |E|$.

(a) Present an algorithm which, given an undirected graph $G = (V, E)$, determines whether G contains a cycle in $O(n)$ time. If the graph has a cycle, it should output the vertices of any cycle.

It is important that the running time of your algorithm is independent of the number of edges m, which may generally be as high as $\Omega(n^2)$. This means that your algorithm will need to terminate with the correct answer, possibly even before reading the entire adjacency list. You may assume that the graph is presented to your algorithm in constant time as reference to its existing adjacency list.

(b) Prove that there is no corresponding algorithm for digraphs. In particular, prove that any algorithm that correctly determines whether a digraph has a cycle may need to inspect $\Omega(n^2)$ edges.

Hint: Prove this by contradiction. Suppose that such an algorithm existed. Show that there exists a digraph having $\Omega(n^2)$ edges, such that any correct algorithm must inspect *every* edge of this graph.

Solution:

(a) The algorithm: Run DFS on the graph (restarting from an undiscovered vertex whenever the current search finishes). If at any point an edge (u, v) other than the edge to u's parent leads to a vertex v that was already discovered, terminate the algorithm. In an undirected graph, every such edge is a back edge, so v is an ancestor of u in the DFS tree. To output the cycle, follow the parent pointers from u up to v; this tree path together with the edge (u, v) forms a cycle.

It is not required that you prove the correctness or runtime of the algorithm, but this algorithm runs in $O(n)$ time because each edge exploration either discovers a new vertex (at most $n - 1$ times), follows the edge back to the parent (at most once per vertex), or terminates the algorithm. Thus there are $O(n)$ edge explorations in total.

If a cycle exists in the graph, the algorithm will find it: if it never terminated early, DFS would explore every edge of the graph, and the last edge of the cycle to be explored would connect to an already-discovered vertex (and is not a parent edge, since a cycle in a simple graph has at least three vertices), terminating the algorithm with a cycle. Thus, every graph with a cycle will be identified by the algorithm.

(b) Consider a digraph with vertices labeled $1, \ldots, n$ and edge set $E = \{(u, v) : u < v\}$. This edge set has size $\binom{n}{2} = \Omega(n^2)$, and the graph has no cycle. However, if any single edge (u, v) with $v < u$ were inserted, the graph would contain a cycle, and reversing any single edge (u, v) with $v \geq u + 2$ also creates a cycle. So an algorithm that has not inspected some edge cannot be certain that the graph is acyclic: an adversary could make that uninspected edge point backwards, and the algorithm would answer incorrectly on the resulting graph. Thus, any correct algorithm must inspect $\Omega(n^2)$ edges before concluding that no cycle exists. This is in contrast to the undirected setting, where any graph with at least n edges must contain a cycle.

Problem 4. In this problem, we seek to characterize graphs that have only one topological ordering. A **Hamiltonian path** is a path that contains every vertex in a graph. Finding a Hamiltonian Path in a general undirected graph is a hard problem, but we will show that it is easier.

(a) Show that if $G = (V, E)$ is a DAG with a Hamiltonian path, then it has only one valid topological ordering.

(b) Show the converse: that if $G = (V, E)$ is a DAG with only one valid topological ordering, then it has a Hamiltonian path.

(c) Provide an algorithm for determining whether a DAG $G = (V, E)$ has a Hamiltonian path. Your algorithm should run in $O(n + m)$ time, where $|V| = n, |E| = m$.

Solution:

(a) Let G be a DAG with a Hamiltonian path $(v_1, v_2, \ldots, v_n)$. Since v_1 has a directed path to every other vertex in the graph, it must be situated at the first spot in any topological ordering. Similarly, v_2, having a path to every other vertex but v_1, must be the second vertex in the ordering. Extending this logic, the only valid topological ordering is that of the vertices in Hamiltonian path order.

(b) Let G be a DAG with only one valid topological ordering, which we denote $(v_1, v_2, \ldots, v_n)$. We aim to show that $(v_1, v_2, \ldots, v_n)$ is a Hamiltonian path, i.e., that $(v_i, v_{i+1}) \in E$ for every $1 \leq i < n$. Fix such an i and consider the ordering $(v_1, \ldots, v_{i-1}, v_{i+1}, v_i, v_{i+2}, \ldots, v_n)$ obtained by exchanging v_i and v_{i+1}. This ordering is invalid by assumption, so some edge must point backwards in it. The only pair whose relative order changed is v_i, v_{i+1}, so that edge must be (v_i, v_{i+1}). Thus each vertex has an edge to the next one in the ordering, and $(v_1, v_2, \ldots, v_n)$ is a Hamiltonian path.

(c) To determine whether a DAG has a Hamiltonian path, run the topological sorting algorithm, which terminates in $O(n + m)$ time. Then, consider the resulting topological ordering $(v_1, \ldots, v_n)$. By parts (a) and (b), there exists a Hamiltonian path if and only if each vertex v_i has an edge to its immediate successor v_{i+1}. Checking this for all i takes $O(n + m)$ time by scanning the adjacency lists, so the total running time is $O(n + m)$.