20.4-5

Another way to topologically sort a directed acyclic graph $G = (V, E)$ is to repeatedly find a vertex of in-degree 0, 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?

20.5 Strongly connected components

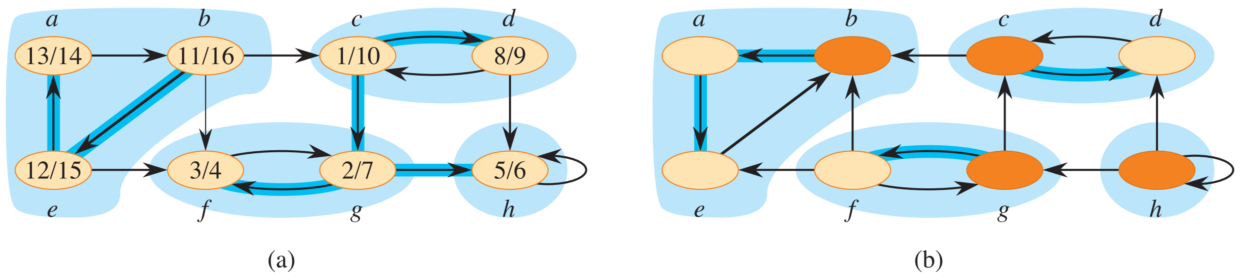
We now consider a classic application of depth-first search: decomposing a directed graph into its strongly connected components. This section shows how to do so using two depth-first searches. Many algorithms that work with directed graphs begin with such a decomposition. After decomposing the graph into strongly connected components, such algorithms run separately on each one and then combine the solutions according to the structure of connections among components.

Recall from Appendix B that a strongly connected component of a directed graph $G = (V, E)$ is a maximal set of vertices $C \subseteq V$ such that for every pair of vertices $u, v \in C$, both $u \rightsquigarrow v$ and $v \rightsquigarrow u$, that is, vertices u and v are reachable from each other. Figure 20.9 shows an example.

The algorithm for finding the strongly connected components of a directed graph $G = (V, E)$ uses the transpose of G, which we defined in Exercise 20.1-3 to be the graph $G^{\mathrm{T}} = (V, E^{\mathrm{T}})$, where $E^{\mathrm{T}} = \{(u, v) : (v, u) \in E\}$. That is, E^{T} consists of the edges of G with their directions reversed. Given an adjacency-list representation of G, the time to create G^{T} is $\Theta(V + E)$. The graphs G and G^{T} have exactly the same strongly connected components: u and v are reachable from each other in G if and only if they are reachable from each other in G^{T}. Figure 20.9(b) shows the transpose of the graph in Figure 20.9(a), with the strongly connected components shaded blue in both parts.

The linear-time (i.e., $\Theta(V + E)$-time) procedure STRONGLY-CONNECTED-COMPONENTS on the next page computes the strongly connected components of a directed graph $G = (V, E)$ using two depth-first searches, one on G and one on G^{T}.

The idea behind this algorithm comes from a key property of the ***component graph*** $G^{\mathrm{SCC}} = (V^{\mathrm{SCC}}, E^{\mathrm{SCC}})$, defined as follows. Suppose that G has strongly connected components $C_1, C_2, \ldots, C_k$. The vertex set V^{SCC} is $\{v_1, v_2, \ldots, v_k\}$, and it contains one vertex v_i for each strongly connected component C_i of G. There is an edge $(v_i, v_j) \in E^{\mathrm{SCC}}$ if G contains a directed edge (x, y) for some $x \in C_i$ and some $y \in C_j$. Looked at another way, if we contract all edges whose incident vertices are within the same strongly connected component of G so that



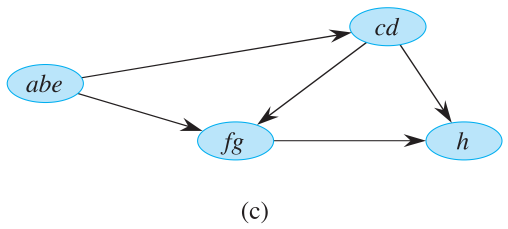


Figure 20.9 **(a)** A directed graph G. Each region shaded light blue is a strongly connected component of G. Each vertex is labeled with its discovery and finish times in a depth-first search, and tree edges are dark blue. **(b)** The graph G^{T}, the transpose of G, with the depth-first forest computed in line 3 of STRONGLY-CONNECTED-COMPONENTS shown and tree edges shaded dark blue. Each strongly connected component corresponds to one depth-first tree. Orange vertices b, c, g, and h are the roots of the depth-first trees produced by the depth-first search of G^{T}. **(c)** The acyclic component graph G^{SCC} obtained by contracting all edges within each strongly connected component of G so that only a single vertex remains in each component.

```
STRONGLY-CONNECTED-COMPONENTS(G)
1  call DFS(G) to compute finish times $u.f$ for each vertex $u$
2  create $G^{\mathrm{T}}$
3  call DFS($G^{\mathrm{T}}$), but in the main loop of DFS, consider the vertices
       in order of decreasing $u.f$ (as computed in line 1)
4  output the vertices of each tree in the depth-first forest formed in line 3 as a
       separate strongly connected component
```

only a single vertex remains, the resulting graph is G^{SCC}. Figure 20.9(c) shows the component graph of the graph in Figure 20.9(a).

The following lemma gives the key property that the component graph is acyclic. We'll see that the algorithm uses this property to visit the vertices of the component graph in topologically sorted order, by considering vertices in the second depth-first search in decreasing order of the finish times that were computed in the first depth-first search.