Strongly connected components

Definition: An undirected graph is connected if every vertex is reachable from all other vertices. That is, for every pair of vertices $u,v \in V$, there is a path from u to v (and therefore, a path from v to u).

Definition: A directed graph G is strongly connected if every two vertices are reachable from each other. That is, for every pair of vertices $u,v \in V$, there is both a path from u to v and a path from v to u.

Definition: A strongly connected component of a directed graph $G = (V,E)$ is a maximal part of the graph that is strongly connected. Formally, it is a maximal set of vertices $C \subseteq V$ such that for every pair of vertices $u,v \in C$, there is both a path from u to v and a path from v to u.

(By "maximal" I mean that no proper superset of the vertices forms a strongly connected component. However, note that a single graph may have multiple strongly connected components of different sizes.)

Definition: A connected component of an undirected graph is a maximal set of vertices where every vertex in the set is reachable from every other vertex in the set. That is, for every pair of vertices u,v in the connected component C, there is a path from u to v.

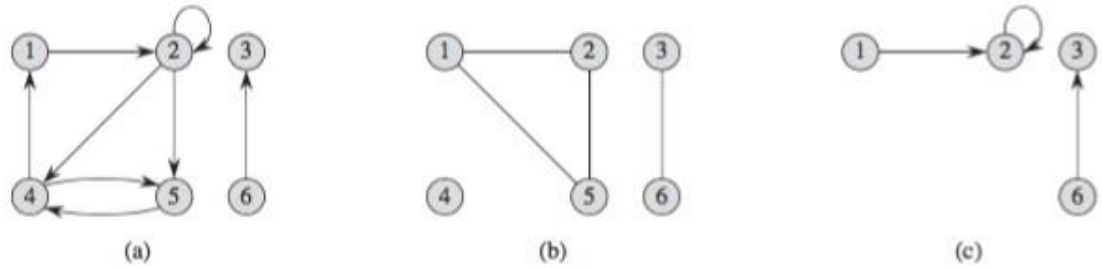

Figure B.2 Directed and undirected graphs. **(a)** A directed graph $G = (V, E)$, where $V = \{1, 2, 3, 4, 5, 6\}$ and $E = \{(1, 2), (2, 2), (2, 4), (2, 5), (4, 1), (4, 5), (5, 4), (6, 3)\}$. The edge $(2, 2)$ is a self-loop. **(b)** An undirected graph $G = (V, E)$, where $V = \{1, 2, 3, 4, 5, 6\}$ and $E = \{(1, 2), (1, 5), (2, 5), (3, 6)\}$. The vertex 4 is isolated. **(c)** The subgraph of the graph in part (a) induced by the vertex set $\{1, 2, 3, 6\}$.

Example: The graph in Figure B.2(a) has three strongly connected components: {1,2,4,5}, {3}, and {6}. The graph in Figure B.2(b) has three connected components: {1,2,5}, {3,6}, and {4}.

If we wanted to find the connected components in an undirected graph, we could simply run depth first search. Each depth first search tree consists of the vertices that are reachable from a given start node, and those nodes are all reachable from each other. However, finding the strongly connected components in a directed graph is not as easy, because u can be reachable from v without v being reachable from u.

2.1 Algorithm for finding strongly connected components

Suppose we decompose a graph into its strongly connected components, and build a new graph, where each strongly connected component is a "giant vertex." Then this new graph forms a directed acyclic graph, as in the figure.

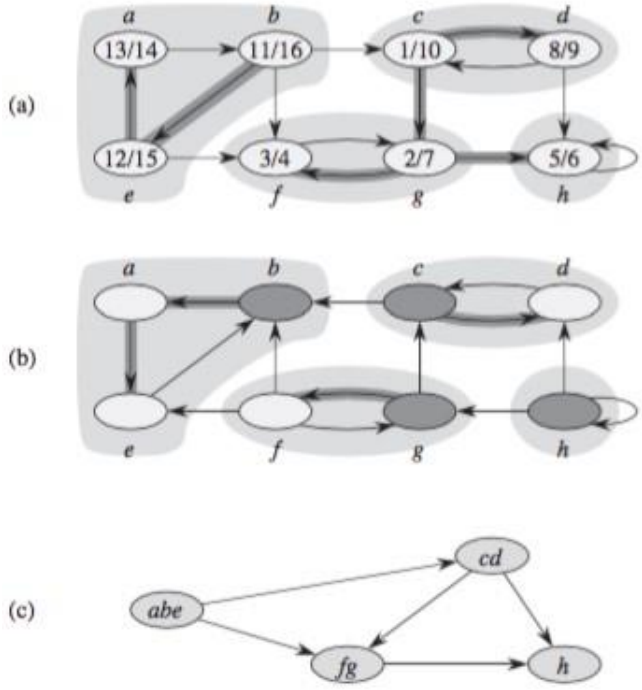

Figure 22.9 **(a)** A directed graph G. Each shaded region is a strongly connected component of G. Each vertex is labeled with its discovery and finishing times in a depth-first search, and tree edges are shaded. **(b)** The graph G^T, the transpose of G, with the depth-first forest computed in line 3 of STRONGLY-CONNECTED-COMPONENTS shown and tree edges shaded. Each strongly connected component corresponds to one depth-first tree. Vertices b, c, g, and h, which are heavily shaded, 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.

To see why the graph must be acyclic, suppose there was a cycle in the strongly connected component graph, $C_{i1} \to C_{i2} \to \cdots \to C_{ik} \to C_{i1}$. Then because

- Any vertex in C_{i1} can be reached from any other vertex in C_{i1}
- Any vertex in C_{i2} can be reached from any other vertex in C_{i2}
- There is an edge from some vertex in C_{i1} to some vertex in C_{i2}

It follows that any vertex in the component C_{i2} can be reached from any other vertex in the component C_{i1}. Similarly, any vertex in any of the components $C_{i1},\ldots,C_{ik}$ can be reached from any other vertex in $C_{i1},\ldots,C_{ik}$. So $C_{i1} \cup \cdots \cup C_{ik}$ is a strongly connected superset of C_{i1}, which violates the assumption that C_{i1} is a maximal strongly connected set of vertices.

2.1.1 Idea

So if we compute the finishing times of each vertex (using depth first search), and then examine the vertices in reverse order of finishing time, we will be examining the components in topologically sorted order. (The vertices themselves can't be topologically sorted, but the strongly connected components can.) Note: this is not obvious, and it's proven in the textbook.

Now suppose we reverse all the edges (which doesn't change the strongly connected components), and perform a (second) depth first search on the new graph, examining the vertices in reverse order of finishing time. Then every time we finish a strongly connected component, the recursion will abruptly halt, because by reversing the edges, we have created a "barrier" between that strongly connected component and the next one. Each depth-first-search tree produces exactly one strongly connected component. We can exploit this to return the strongly connected components.

2.1.2 Algorithm

STRONGLY-CONNECTED-COMPONENTS(G)
1 call DFS(G) to compute finishing times $u.f$ for each vertex u
2 compute G^T
3 call DFS(G^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

2.1.3 Correctness

We are just proving a lemma here. For the full correctness proof, see the textbook.

If C is a component, we let $d(C) = \min_{u \in C}(u.d)$ be the earliest discovery time out of any vertex in C, and $f(C) = \max_{u \in C}(u.f)$ be the latest finishing time out of any vertex in C.

Lemma 22.14: Let C and C^0 be distinct strongly connected components in a directed graph $G = (V, E)$. Suppose there is an edge $(u,v) \in E$, where $u \in C$ and $v \in C^0$. Then $f(C) > f(C^0)$.

This lemma implies that if vertices are visited in reverse order of finishing time, then the components will be visited in topologically sorted order. That is, if there is an edge from component 1 to component 2, then component 1 will be visited before component 2.

Proof: Consider two cases.

1. $d(C) < d(C^0)$ (i.e. component C was discovered first). Let x be the first vertex discovered in C. Then at time $x.d$, all vertices in C and C^0 are white, so there is a path from x to each vertex in C consisting only of white vertices. There is also a path from

x to each vertex in C^0 consisting of only white vertices (because you can follow the edge (u,v)). By the white path theorem, all vertices in C and C^0 become descendants of x in the depth first search tree.

Now x has the latest finishing time out of any of its descendants, due to the "parenthesis theorem." However, you can see this if you look at the structure of the recursive calls – we call DFS-Visit on x, and then recurse on the nodes that will become its descendants in the depth-first-search tree. So the inner recursion has to finish before the outer recursion, which means x has the latest finishing time out of all the nodes in C and C^0. Therefore, $x.f = f(C) > f(C^0)$.

2. $d(C) > d(C^0)$ (i.e. component C^0 was discovered first). Let y be the first vertex discovered in C^0. At time $y.d$, all vertices in C^0 are white, and there is a path from y to each vertex in C^0 consisting only of white vertices. By the white path theorem, all vertices in C^0 are descendants of y in the depth-first-search tree, so $y.f = f(C^0)$.

However, there is no path from C^0 to C (because we already have an edge from C to C^0, and if there was a path in the other direction, then C and C^0 would just be one component). Now all of the vertices in C are white at time $y.d$ (because $d(C) > d(C^0)$). And since no vertex in C is reachable from y, all the vertices in C must still be white at time $y.f$. Therefore, the finishing time of any vertex in C is greater than $y.f$, and $f(C) > f(C^0)$.