CSCI 470: Graphs, Depth-First-Search, Topological Sort

Vijay Chaudhary

October 18, 2023

Department of Electrical Engineering and Computer Science
Howard University

Overview

1. DFS
2. DFS properties
3. Topological sort
4. Strongly connected components
5. Breadth-First Search

DFS

Depth-First Search

- The strategy followed by depth-first search is, as its name implies, to search “deeper” in the graph whenever possible.

Depth-First Search

- The strategy followed by depth-first search is, as its name implies, to search “deeper” in the graph whenever possible.
- Depth-first search explores edges out of the most recently discovered vertex v that still has unexplored edges leaving it.

Depth-First Search

- The strategy followed by depth-first search is, as its name implies, to search “deeper” in the graph whenever possible.
- Depth-first search explores edges out of the most recently discovered vertex v that still has unexplored edges leaving it.
- Once all of v’s edges have been explored, the search “backtracks” to explore edges leaving the vertex from which v was discovered.

Depth-First Search

- The strategy followed by depth-first search is, as its name implies, to search “deeper” in the graph whenever possible.
- Depth-first search explores edges out of the most recently discovered vertex v that still has unexplored edges leaving it.
- Once all of v’s edges have been explored, the search “backtracks” to explore edges leaving the vertex from which v was discovered.
- This process continues until we have discovered all the vertices that are reachable from the original source vertex.

Depth-First Search

- The strategy followed by depth-first search is, as its name implies, to search "deeper" in the graph whenever possible.
- Depth-first search explores edges out of the most recently discovered vertex v that still has unexplored edges leaving it.
- Once all of v's edges have been explored, the search "backtracks" to explore edges leaving the vertex from which v was discovered.
- This process continues until we have discovered all the vertices that are reachable from the original source vertex.
- If any undiscovered vertices remain, then depth-first search selects one of them as a new source, and it repeats the search from that source. The algorithm repeats this entire process until it has discovered every vertex.

DFS: pseudocode

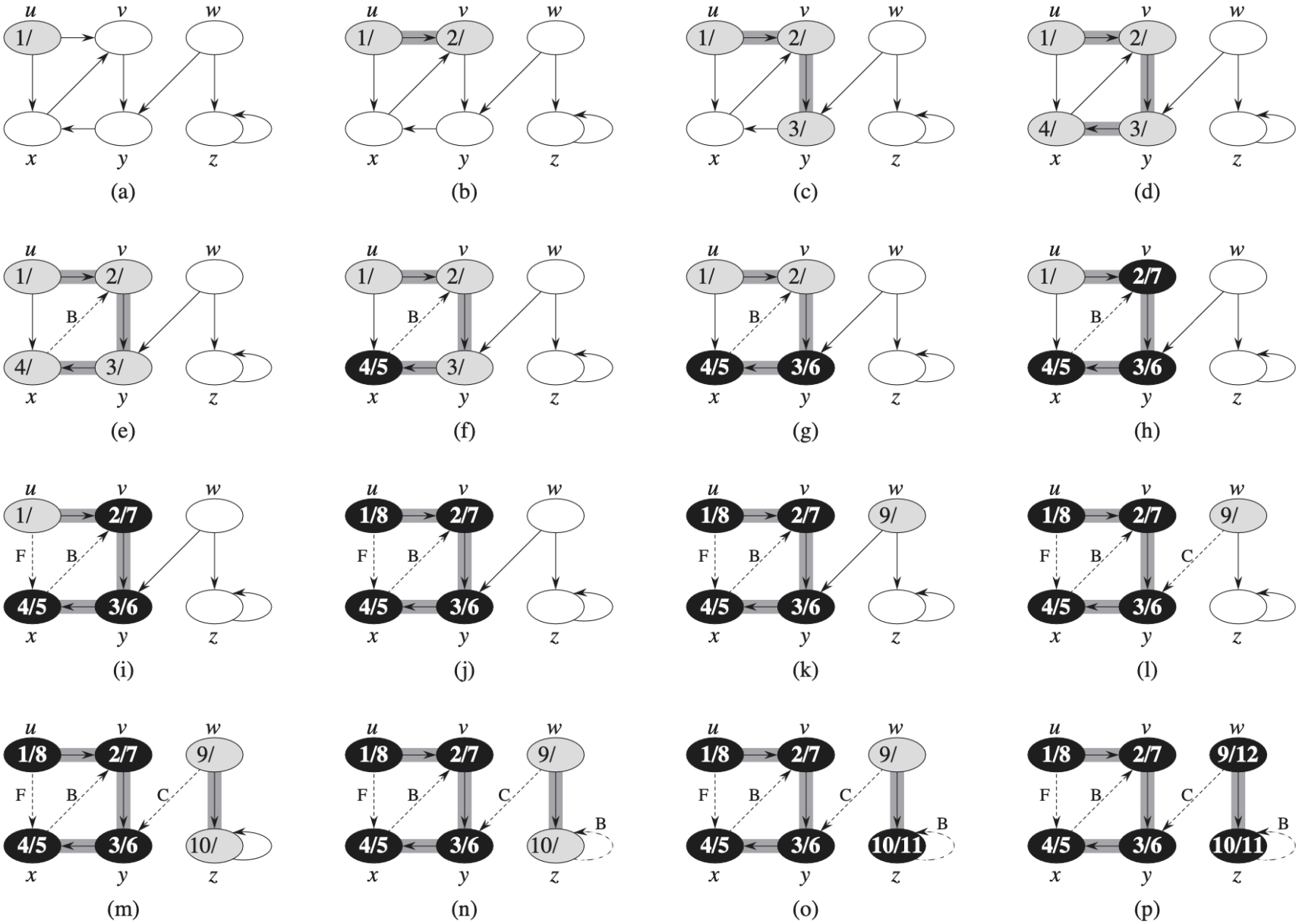
```
DFS(G)
1  for each vertex u ∈ G.V
2        u.color = WHITE
3        u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6        if u.color == WHITE
7              DFS-VISIT(G, u)
```

DFS-Visit: pseudocode

DFS-VISIT(G, u)

```
 1  time = time + 1 // white vertex u has just been discovered
 2  u.d = time
 3  u.color = GRAY
 4  for each v ∈ G.Adj[u] // explore edge (u, v)
 5        if v.color == WHITE
 6              v.π = u
 7              DFS-VISIT(G, v)
 8  u.color = BLACK // blacken u; it is finished
 9  time = time + 1
10  u.f = time
```

Illustration

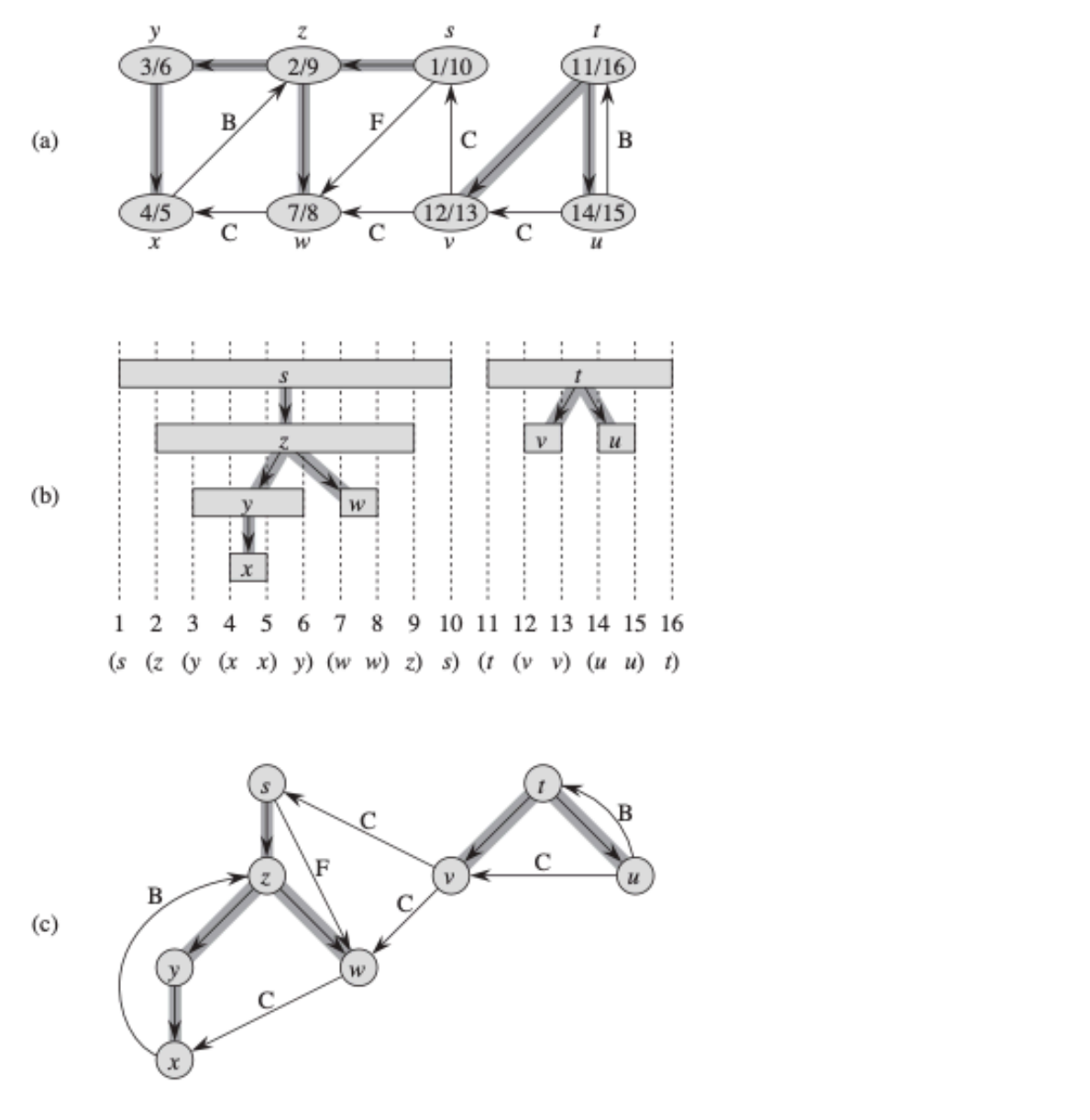


DFS properties

Properties of depth-first search

1. Vertex v is a descendent of vertex u in the depth-first forest if and only if v is discovered during the time that u is gray.
2. **Parenthesis structure**: If v is a descendent of u in the depth-first forest, then $[v.d, v.f]$ is contained entirely within $[u.d, u.f]$. If neither vertex is a descendent of the other, then the intervals are disjoint.
3. **White path theorem**: In a depth-first forest, v is a descendent of u if and only if at the time $u.d$ that the search discovers u, there is a path from u to v consisting entirely of white vertices.

Properties of DFS



Property of DFS

- Vertex v is a descendent of vertex u in the depth-first forest if and only if v is discovered during the time that u is gray.

Parenthesis structure

In any depth-first search of a (directed or undirected) graph $G = (V, E)$, for any two vertices u and v, exactly one of the following three conditions holds:

- the intervals $[u.d, u.f]$ and $[v.d, v.f]$ are entirely disjoint, and neither u nor v is a descendant of the other in the depth-first forest,

Parenthesis structure

In any depth-first search of a (directed or undirected) graph $G = (V, E)$, for any two vertices u and v, exactly one of the following three conditions holds:

- the intervals $[u.d, u.f]$ and $[v.d, v.f]$ are entirely disjoint, and neither u nor v is a descendant of the other in the depth-first forest,
- the interval $[u.d, u.f]$ is contained entirely within the interval $[v.d, v.f]$, and u is a descendant of v in a depth-first tree, or

Parenthesis structure

In any depth-first search of a (directed or undirected) graph $G = (V, E)$, for any two vertices u and v, exactly one of the following three conditions holds:

- the intervals $[u.d, u.f]$ and $[v.d, v.f]$ are entirely disjoint, and neither u nor v is a descendant of the other in the depth-first forest,
- the interval $[u.d, u.f]$ is contained entirely within the interval $[v.d, v.f]$, and u is a descendant of v in a depth-first tree, or
- the interval $[v.d, v.f]$ is contained entirely within the interval $[u.d, u.f]$, and v is a descendant of u in a depth-first tree.

Parenthesis structure: Proof

We begin with the case in which $u.d < v.d$. We consider two subcases, according to whether $v.d < u.f$ or not.

1. The first case occurs when $v.d < u.f$, so v was discovered while u was still gray, which implies that v is a descendant of u. Moreover, since v was discovered more recently than u, all of its outgoing edges are explored, and v is finished, before the search returns to and finishes u. In this case, therefore, the interval $[v.d, v.f]$ is entirely contained within the interval $[u.d, u.f]$.

Parenthesis structure: Proof

We begin with the case in which $u.d < v.d$. We consider two subcases, according to whether $v.d < u.f$ or not.

1. The first case occurs when $v.d < u.f$, so v was discovered while u was still gray, which implies that v is a descendant of u. Moreover, since v was discovered more recently than u, all of its outgoing edges are explored, and v is finished, before the search returns to and finishes u. In this case, therefore, the interval $[v.d, v.f]$ is entirely contained within the interval $[u.d, u.f]$.
2. In the other subcase, $u.f < v.d$, and $u.d < u.f$, $u.d < u.f < v.d < v.f$; thus the intervals $[u.d, u.f]$ and $[v.d, v.f]$ are disjoint. Because the intervals are disjoint, neither vertex was discovered while the other was gray, and so neither vertex is a descendant of the other.

Parenthesis structure: Proof

We begin with the case in which $u.d < v.d$. We consider two subcases, according to whether $v.d < u.f$ or not.

1. The first case occurs when $v.d < u.f$, so v was discovered while u was still gray, which implies that v is a descendant of u. Moreover, since v was discovered more recently than u, all of its outgoing edges are explored, and v is finished, before the search returns to and finishes u. In this case, therefore, the interval $[v.d, v.f]$ is entirely contained within the interval $[u.d, u.f]$.
2. In the other subcase, $u.f < v.d$, and $u.d < u.f$, $u.d < u.f < v.d < v.f$; thus the intervals $[u.d, u.f]$ and $[v.d, v.f]$ are disjoint. Because the intervals are disjoint, neither vertex was discovered while the other was gray, and so neither vertex is a descendant of the other.
    - The case in which $v.d < u.d$ is similar, with the roles u and v reversed in the above argument. □

Corollary 22.8 (Nesting of descendant's intervals)

Corollary: Vertex v is a proper descendant of vertex u in the depth-first search forest for a (directed or undirected) graph G if and only if $u.d < v.d < v.f < u.f$.
This can immediately be shown using the first case in the previous proof.

White path theorem

In a depth-first forest, v is a descendent of u if and only if at the time $u.d$ that the search discovers u, there is a path from u to v consisting entirely of white vertices.

White path theorem: proof ($\Rightarrow$)

$\Rightarrow$:

- If $v = u$, then the path from u to v contains just vertex u, which is still white when we set the value of $u.d$.

White path theorem: proof ($\Rightarrow$)

$\Rightarrow$:

- If $v = u$, then the path from u to v contains just vertex u, which is still white when we set the value of $u.d$.
- Now, suppose that v is a proper descendent of u in the depth-first forest. By Corollary 22.8, $u.d < v.d$, and so v is white at the time $u.d$. Since v can be any descendant of u, all vertices on the unique simple path from u to v in the depth-first forest are white at time $u.d$.

White path theorem: proof ($\Leftarrow$)

$\Leftarrow$:

- Suppose that there is a path of white vertices from u to v at time $u.d$, but v does not become a descendant of u in the depth-first tree.

White path theorem: proof ($\Leftarrow$)

$\Leftarrow$:

- Suppose that there is a path of white vertices from u to v at time $u.d$, but v does not become a descendant of u in the depth-first tree.
- Without loss of generality, assume that every vertex other than v along the path becomes a descendant of u. (Otherwise, let v be the closest vertex to u along the path that doesn't become a descendant of u.)

White path theorem: proof ($\Leftarrow$)

$\Leftarrow$:

- Suppose that there is a path of white vertices from u to v at time $u.d$, but v does not become a descendant of u in the depth-first tree.
- Without loss of generality, assume that every vertex other than v along the path becomes a descendant of u. (Otherwise, let v be the closest vertex to u along the path that doesn't become a descendant of u.)
- Let w be the predecessor of v in the path, so that w is a descendant of u (w and u may in fact be the same vertex).

White path theorem: proof ($\Leftarrow$)

$\Leftarrow$:

- Suppose that there is a path of white vertices from u to v at time $u.d$, but v does not become a descendant of u in the depth-first tree.
- Without loss of generality, assume that every vertex other than v along the path becomes a descendant of u. (Otherwise, let v be the closest vertex to u along the path that doesn't become a descendant of u.)
- Let w be the predecessor of v in the path, so that w is a descendant of u (w and u may in fact be the same vertex).
- By Corollary 22.8, $w.f \leq u.f$. Because v must be discovered after u is discovered, but before w is finished, we have $u.d < v.d < w.f \leq u.f$. Theorem 22.7 then implies that the interval $[v.d, v.f]$ is contained entirely within the interval $[u.d, u.f]$. By Corollary 22.8, v must after all be a descendant of u. $\square$

Classification of edges

- ***Tree edges*** are edges in the depth-first forest G_π. Edge (u, v) is a tree edge if v was first discovered by exploring edge (u, v).
- ***Back edges*** are edges (u, v) connecting a vertex u to an ancestor v in a depth-first tree. We consider self-loops, which may occur in directed graphs, to be back edges.
- ***Forward edges*** are those nontree edges (u, v) connecting a vertex u to be a descendant v in a depth-first tree. Non-tree edge is an edge leading to a marked vertex.
- ***Cross edges*** are all other edges. They can go between vertices in the same depth-first tree, as long as one vertex is not an ancestor of the other, or they can go between vertices in different depth-first trees. Alternatively, Cross edges point from a node to a previously visited node that is neither an ancestor nor a descendant.

Topological sort

Topological sort

- A ***toplogical sort*** of a dag $G = (V, E)$ is a linear ordering of all its vertices such that if G contains an edge (u, v), then u appears before v in the ordering.
- If a graph contains a cycle, then no linear order is possible.
- We can view a topological sort of a graph as an ordering of its vertices along a horizontal line so that all directed edges go from left to right.
- Topological sorting is thus different from the usual kind of "sorting" as compared to what we did in previous lectures.

Topological sort: pseudocode

TOPOLOGICAL-SORT(G)

1 call DFS(G) to compute finishing times $v.f$ for each vertex v
2 as each vertex is finished, insert it onto the front of a linked list
3 **return** the linked list of vertices

- We can perform a topological sort in time $\Theta(V + E)$, since depth-first search takes $O(1)$ time to insert each of the $|V|$ vertices onto the front of the linked list.

Topological sort: Figure

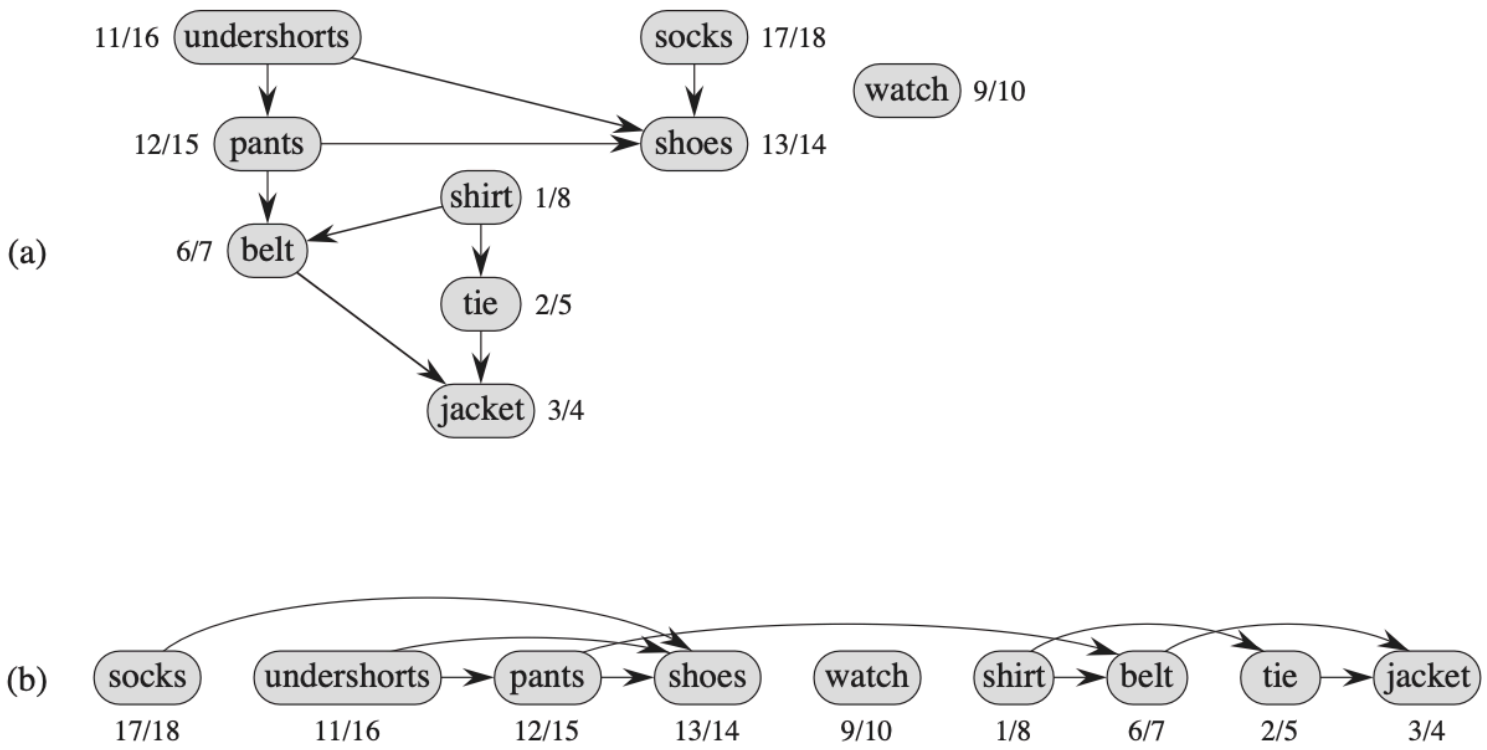


Figure 22.7 **(a)** Professor Bumstead topologically sorts his clothing when getting dressed. Each directed edge (u, v) means that garment u must be put on before garment v. The discovery and finishing times from a depth-first search are shown next to each vertex. **(b)** The same graph shown topologically sorted, with its vertices arranged from left to right in order of decreasing finishing time. All directed edges go from left to right.

Topological sort: Lemma 22.11

Lemma 22.11 A directed graph G is a acyclic if and only if a depth-first search of G yields no back edges.

Lemma 22.11: Proof ($\Rightarrow$)

$\Rightarrow$

- Suppose that a depth-first search produces a back edge (u, v). Then vertex v is an ancestor of vertex u in the depth-first forest. Thus, G contains a path from v to u, and the back edge (u, v) completes a cycle.

Lemma 22.11: Proof ($\Leftarrow$)

$\Leftarrow$

- Suppose that G contains a cycle c. We show that a depth-first search of G yields a back edge.
- Let v be the first vertex to be discovered in c, and let (u, v) be the preceding edge in c. At time $v.d$, the vertices of c form a path white vertices from v to u. By the white-path theorem, vertex u becomes a descendant of v in the depth-first forest. Therefore, (u, v) is a back edge.

Topological sort: Theorem 22.12

TOPOLOGICAL-SORT produces a topological sort of the directed acyclic graph provides as its input.

- Suppose that DFS is run on a given dag $G = (V, E)$ to determine finishing times for its vertices. It suffices to show that for any pair of distinct vertices $u, v \in V$, if G contains an edge from u to v, then $v.f < u.f$.

Topological sort: Theorem 22.12

TOPOLOGICAL-SORT produces a topological sort of the directed acyclic graph provides as its input.

- Suppose that DFS is run on a given dag $G = (V, E)$ to determine finishing times for its vertices. It suffices to show that for any pair of distinct vertices $u, v \in V$, if G contains an edge from u to v, then $v.f < u.f$.
- Consider any edge (u, v) explored in DFS(G). When this edge is explored, v cannot be gray, since then v would be an ancestor of u and (u, v) would be a back edge, contradicting Lemma 22.11. Therefore, v must be either white or black.

Topological sort: Theorem 22.12

TOPOLOGICAL-SORT produces a topological sort of the directed acyclic graph provides as its input.

- Suppose that DFS is run on a given dag $G = (V, E)$ to determine finishing times for its vertices. It suffices to show that for any pair of distinct vertices $u, v \in V$, if G contains an edge from u to v, then $v.f < u.f$.
- Consider any edge (u, v) explored in DFS(G). When this edge is explored, v cannot be gray, since then v would be an ancestor of u and (u, v) would be a back edge, contradicting Lemma 22.11. Therefore, v must be either white or black.
- If v is white, it becomes a descendant of u, and so $v.f < u.f$. If v is black, it has already been finished, so that $v.f$ has already been set.

Topological sort: Theorem 22.12

TOPOLOGICAL-SORT produces a topological sort of the directed acyclic graph provides as its input.

- Suppose that DFS is run on a given dag $G = (V, E)$ to determine finishing times for its vertices. It suffices to show that for any pair of distinct vertices $u, v \in V$, if G contains an edge from u to v, then $v.f < u.f$.
- Consider any edge (u, v) explored in DFS(G). When this edge is explored, v cannot be gray, since then v would be an ancestor of u and (u, v) would be a back edge, contradicting Lemma 22.11. Therefore, v must be either white or black.
- If v is white, it becomes a descendant of u, and so $v.f < u.f$. If v is black, it has already been finished, so that $v.f$ has already been set.
- Because we are still exploring from u, we have yet to assign a timestamp to $u.f$, and so once we do, we will have $v.f < u.f$ as well. Thus, for any edge (u, v) in the dag, we have $v.f < u.f$, proving the theorem.

Strongly connected components

Strongly connected components

- 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).

Strongly connected components

- 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).
- 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.

Strongly connected components

- 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.
- A connected component of an undirected graph is a maximal set of vertices where every vertex in the set is reachable from every 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.

SCC: Figure

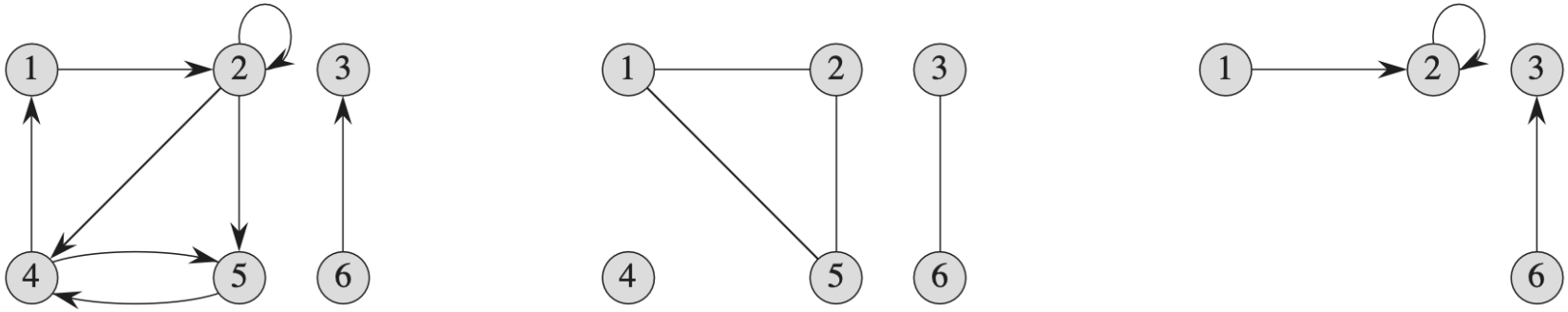


Figure B.2 Directed and undirected graphs. **(a)** A directed graph $G = (V, E)$, where $V = \{1, 2, 3, 4, 5, 6\}$ and $E = \{(1, 2), (2, 2), (2, 4), (2, 5), (4, 1), (4, 5), (5, 4), (6, 3)\}$. The edge $(2, 2)$ is a self-loop. **(b)** An undirected graph $G = (V, E)$, where $V = \{1, 2, 3, 4, 5, 6\}$ 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 $\{1, 2, 3, 6\}$.

SCC

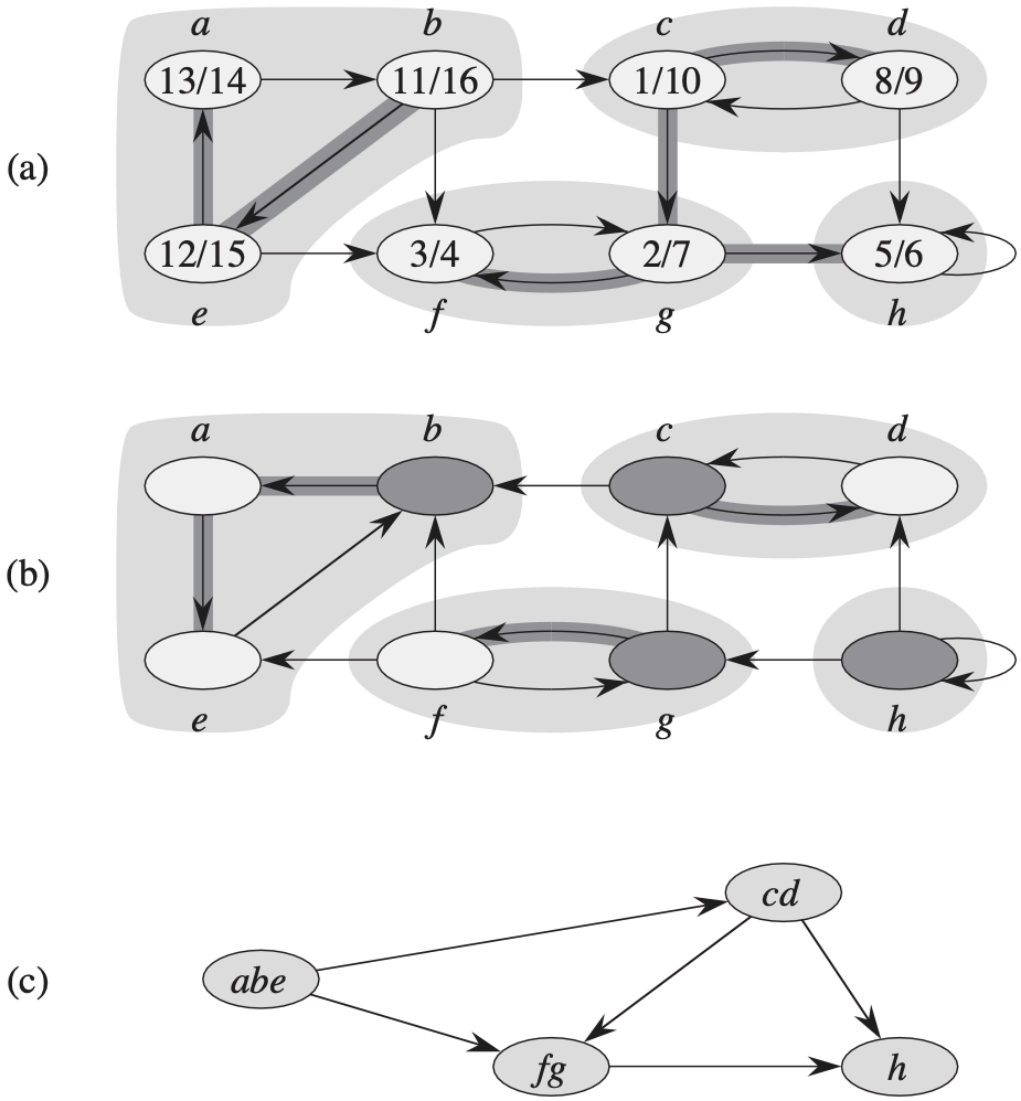
- Our algorithm for finding strongly connected components of a graph $G = (V, E)$ uses the transpose of G.
- We define $G^T = (V, E^T)$, where $E^T = \{(u, v) : (v, u) \in E\}$. That is, E^T consists of the edges of G with their directions reversed.
- It is interesting to observe that G and G^T have 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.

SCC: pesudocode

STRONGLY-CONNECTED-COMPONENTS(G)

1 call DFS to compute finishing times $u.f$ for each vertex u
2 compute G^T
3 call DFS(G^T), but in the main loop 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 separate strongly connected component

SCC: Figure


(a)
a
b
c
d
13/14
11/16
1/10
8/9
12/15
3/4
2/7
5/6
e
f
g
h
(b)
a
b
c
d
e
f
g
h
(c)
cd
abe
fg
h

Breadth-First Search

BFS