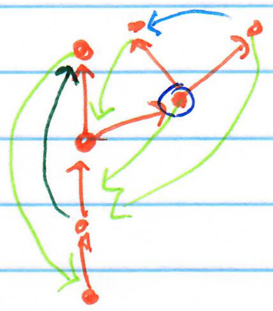
Depth First Search

Edges GRAY → { GRAY — OLD, UNFINISHED VERTEX — ANCESTOR — BACK EDGE

WHITE — NEW VERTEX — TREE EDGE

BLACK — DEAD — COUSIN, AUNT / UNCLE — CROSS EDGES; FORWARD }



u, v

d(u) d(v)

f(u) f(v)

PARENTHESIS THEOREM

$$\begin{cases} \overset{(}{d(u)} \le \overset{[}{d(v)} \le \overset{)}{f(u)} \le \overset{]}{f(v)} & \text{NO} \\ \overset{(}{d(u)} \le \overset{)}{f(u)} \le \overset{[}{d(v)} \le \overset{]}{f(v)} & \text{OK} \\ \overset{(}{d(u)} \le \overset{[}{d(v)} \le \overset{]}{f(v)} \le \overset{)}{f(u)} & \text{OK} \end{cases}$$

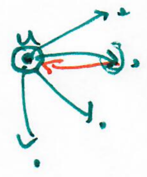
~~d(u) ≤ f(u) ≤ d(v) ≤ f(v)~~

```
function DFS($G$)
1:  for all $u \in V[G]$ do
2:    $color[u] \leftarrow$ WHITE
3:    $\pi[u] \leftarrow$ NIL
4:  end for
5:  $time \leftarrow 0$
6:  for all $u \in V[G]$ do
7:    if $color[u] =$ WHITE then
8:      DFS-visit(u)
9:    end if
10: end for
```

$u \rightarrow v$
- tree
- backedge
- cross edge
- forward edge

```
function DFS-visit($u$)
1:  $color[u] \leftarrow$ GRAY
2:  $d[u] \leftarrow time \leftarrow time + 1$
3:  for all $\boxed{v \in Adj[u]}$ do
4:    if $color[v] =$ WHITE then
5:      $\pi[v] \leftarrow u$        PREDECESSOR
6:      DFS-visit($v$)
7:    end if
8:  end for
9:  $color[u] \leftarrow$ BLACK
10: $f[u] \leftarrow time \leftarrow time + 1$
```

Mark edge w/type



$d \leftarrow time{+}{+}$

Time?

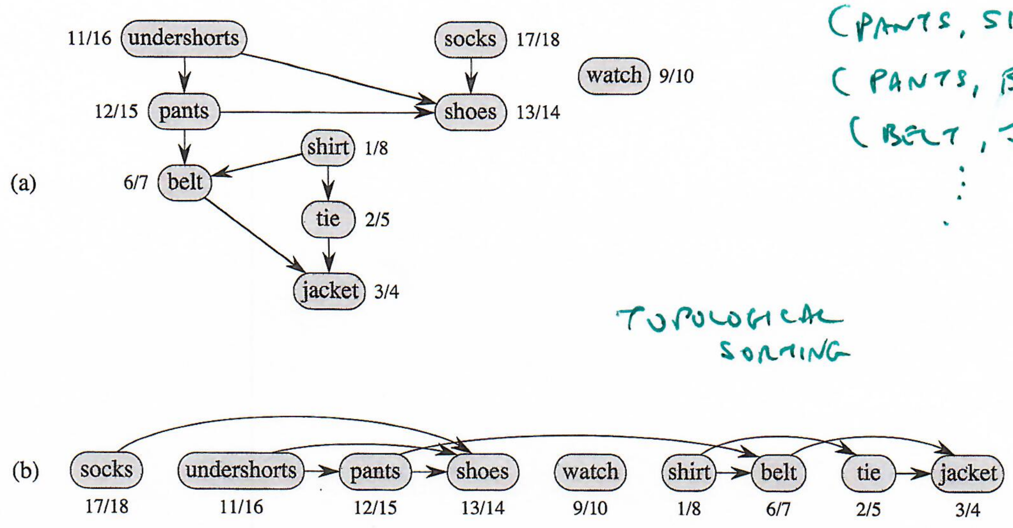
$G = (V, E)$

$O(|E|)$ $O(|V|)$

$O(|V| + |E|)$

3

(PANTS, SHOES)
(PANTS, BELT)
(BELT, JACKET)
⋮



TOPOLOGICAL SORTING

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.

pants). A directed edge (u, v) in the dag of Figure 22.7(a) indicates that garment u must be donned before garment v. A topological sort of this dag therefore gives an order for getting dressed. Figure 22.7(b) shows the topologically sorted dag as an ordering of vertices along a horizontal line such that all directed edges go from left to right.

The following simple algorithm topologically sorts a dag:

```
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
```

Figure 22.7(b) shows how the topologically sorted vertices appear in reverse order of their finishing times.

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

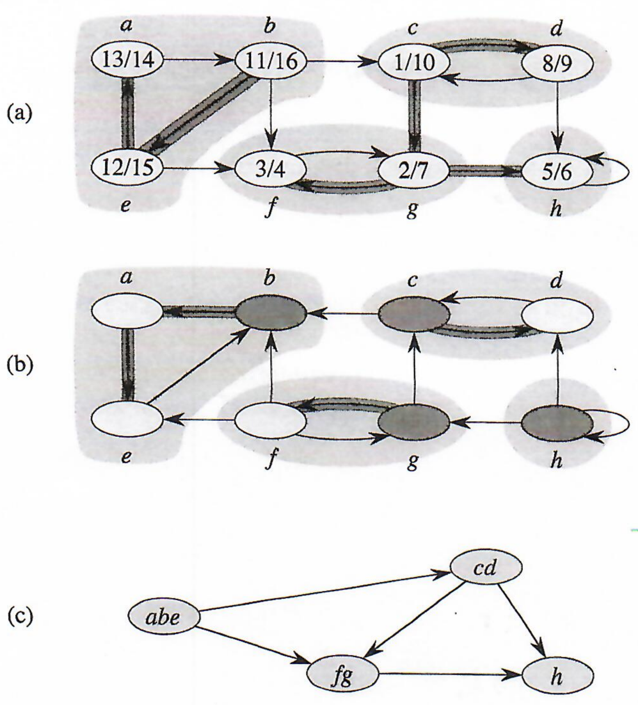
We prove the correctness of this algorithm using the following key lemma characterizing directed acyclic graphs.

STRONGLY CONNECTED COMPONENTS

G

G^T

Schematic G



Schematic

Topologically sorted order of G^T

decreasing order by finishing time

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.

Our algorithm for finding strongly connected components of a graph $G = (V, E)$ uses the transpose of G, which we defined in Exercise 22.1-3 to be the graph $G^T = (V, E^T)$, where $E^T = \{(u, v) : (v, u) \in E\}$. That is, E^T consists of the edges of G with their directions reversed. Given an adjacency-list representation of G, the time to create G^T is $O(V + E)$. 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. Figure 22.9(b) shows the transpose of the graph in Figure 22.9(a), with the strongly connected components shaded.

5

The following linear-time (i.e., $\Theta(V+E)$-time) algorithm 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}.

```
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
```

in topological sort order

The idea behind this algorithm comes from a key property of the ***component graph*** $G^{\mathrm{SCC}} = (V^{\mathrm{SCC}}, E^{\mathrm{SCC}})$, which we define 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 a 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, by contracting all edges whose incident vertices are within the same strongly connected component of G, the resulting graph is G^{SCC}. Figure 22.9(c) shows the component graph of the graph in Figure 22.9(a).

The key property is that the component graph is a dag, which the following lemma implies.

Lemma 22.13
Let C and C' be distinct strongly connected components in directed graph $G = (V, E)$, let $u, v \in C$, let $u', v' \in C'$, and suppose that G contains a path $u \leadsto u'$. Then G cannot also contain a path $v' \leadsto v$.

Proof If G contains a path $v' \leadsto v$, then it contains paths $u \leadsto u' \leadsto v'$ and $v' \leadsto v \leadsto u$. Thus, u and v' are reachable from each other, thereby contradicting the assumption that C and C' are distinct strongly connected components. ■

We shall see that by considering vertices in the second depth-first search in decreasing order of the finishing times that were computed in the first depth-first search, we are, in essence, visiting the vertices of the component graph (each of which corresponds to a strongly connected component of G) in topologically sorted order.

Because the STRONGLY-CONNECTED-COMPONENTS procedure performs two depth-first searches, there is the potential for ambiguity when we discuss $u.d$ or $u.f$. In this section, these values always refer to the discovery and finishing times as computed by the first call of DFS, in line 1.