Graphs Part 3: applications of DFS

Application 1 DFS: topological sort:

A directed edge from u to v, means that v happens after u. Topological order: All directed edges only go forward (needs to be acyclic graph). When we have tasks that one needs to be done before another. Again linear time in vertices and edges O(n+m), since doing DFS.

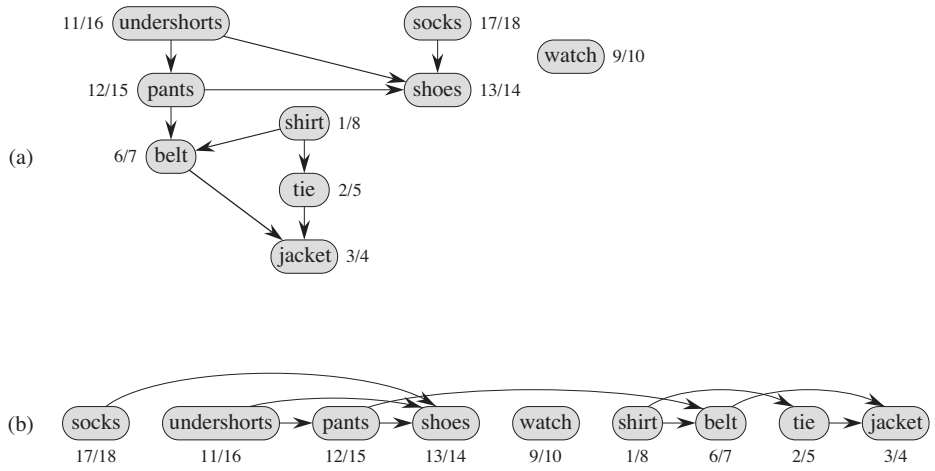

Figure 22.7 **(a)** Professor Bumstead topologically sorts his clothing when getting dressed. Each directed edge (u, v) means that garment u must be put on before garment v. The discovery and finishing times from a depth-first search are shown next to each vertex. **(b)** The same graph shown topologically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

Pseudo code: Topological Sort (G)

1. Call DFS(G) to compute finish times v.f for each vertex v
2. As each vertex is finished, insert into the front of a linked list
3. return the linked **list of vertices**

Application 2 DFS: Strongly Connected Components

This is another example application of Depth First Search (DFS).

Definition: Maximal set of vertices for a directed graph, so that for each pair of vertices, u and v, there is a path both from u to v and from v to u.

Usefulness: Groups of people or populations that are connected amongst themselves; brain regions that are strongly connected; figuring out if a network such as telephone lines are strongly connected (you can dial anyone in the network); Web connectivity.

We will use Depth First Search (DFS), but not just by itself.

Consider panel (a) in the figure below (we will look at panels b and c afterwards):

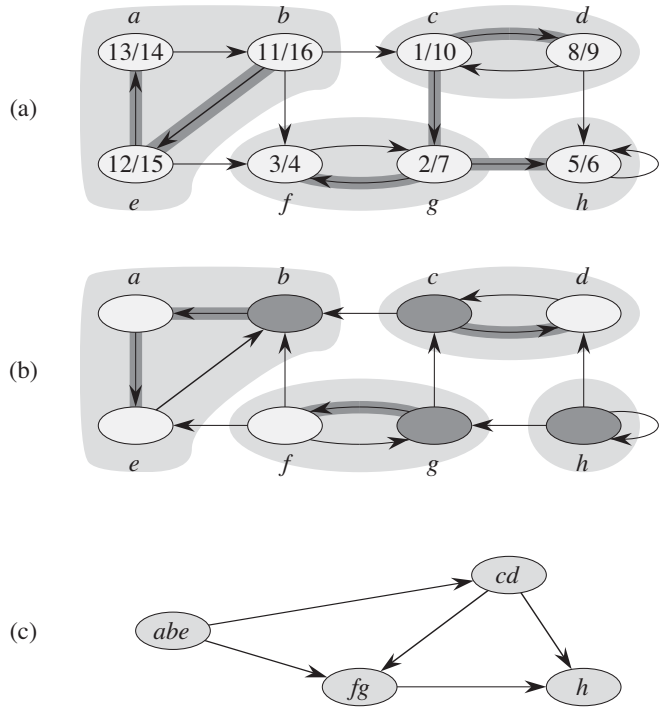

Figure 22.9 **(a)** A directed graph G. Each shaded region is a strongly connected component of G. Each vertex is labeled with its discovery and finishing times in a depth-first search, and tree edges are shaded. **(b)** The graph G^{T}, the transpose of G, with the depth-first forest computed in line 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.

Each cluster of strongly connected components is in gray. For example, there is a path from node c to d, and from node d to c. There is also a path from any node in a,b,e to other nodes in that cluster. If we started DFS from node c, we could discover strongly connected component cluster c and d. But if we started the DFS search from node a, we will actually discover all the nodes of this graph, and not an individual strongly connected component cluster (because b has an arrow out of the cluster going to c, so c would be discovered too, although it is not strongly connected with a,b,e).

Solution: We need to call DFS from the right places to find the strongly connected components!

Algorithm:

1. Call DFS on graph G to compute finish times u.f for each vertex u in graph G (this is step a in the figure above)

2. Compute the transpose of graph G: G^T

This is the graph G, but with all edge directions reversed (see panel b in figure above). Note that G and its transpose have the same strongly connected component clusters.

3. Call DFS on the transpose graph, but in the DFS loop consider vertices in decreasing order of finish time from the DFS we ran on G (starting from highest finish time). This process is also shown in panel b. Since we are considering vertices in decreasing order of finish times from the DFS in panel (a), we first consider vertex b and perform DFS in panel (b) on the transposed graph. We find strongly connected components b,a,e in this way. From this cluster of 3 vertices, there are no are no more vertices going out in the transpose graph. So as in DFS, we choose a new starting node, and again take the largest remaining finish time from panel (a) and therefore start in panel (b) from node c. We find nodes c and d in this way. Again, there are no more nodes going out of the cluster. We start from g, and find strongly connected component cluster g,f. Finally, we also start from vertex h and find only the vertex itself as the last strongly connected component (with itself).

4. Panel c shows the resulting strongly connected components, with each cluster written out as its own vertex. This also shows the idea of strongly connected components. Each cluster has a path between all nodes in the cluster, but between clusters there is only a single arrow (otherwise if there were arrows going both ways, both clusters would form a larger strongly connected component). You could also see that the strongly connected components of graph G and its transpose are the same (just the arrow between strongly connected components is reversed in the transpose graph).

Why it works: The main intuition is that the finish times in the original DFS on graph G, give us the proper starting points for our second DFS, such that we each time discover a strongly connected component cluster and there are no arrows to other undiscovered clusters. For example, the first cluster b,a,e in panel (b), had no out going arrows from these vertices. The second cluster c,d only had out going arrows to the cluster that was already discovered. And so on. There is a lemma and proof in the book, which we did not go over.