Topologically Sorting a Directed Acyclic Graph

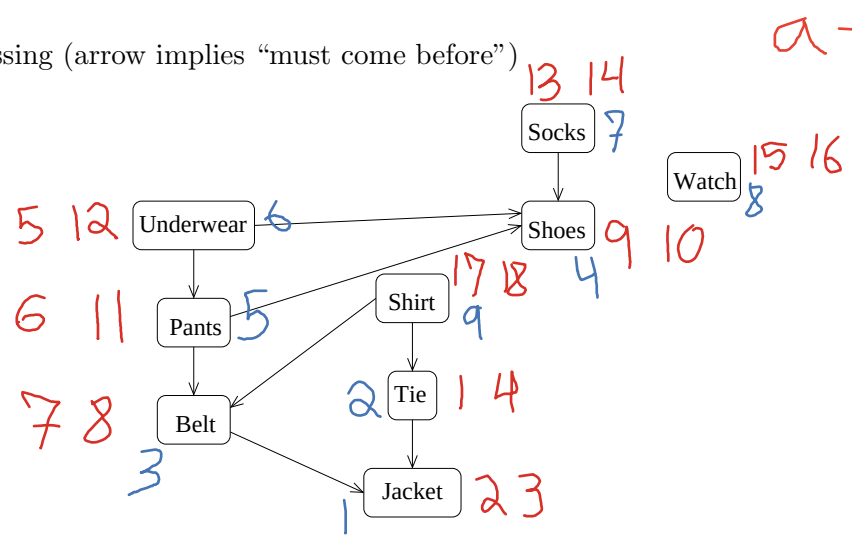
(CLRS 22.4)

1 The problem

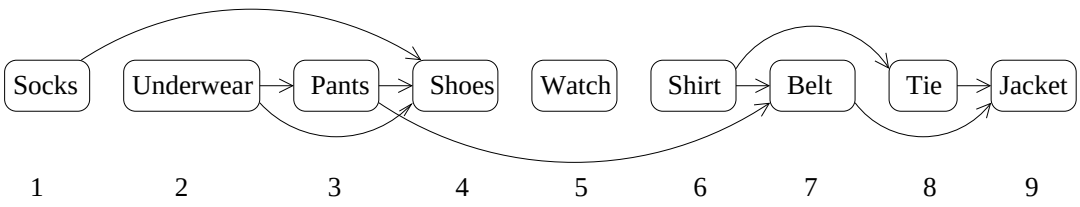
- A topological sorting of a *directed acyclic graph* $G = (V, E)$ is a linear ordering of vertices V such that $(u, v) \in E \Rightarrow u$ appear before v in ordering.

  Intuitively, we think of an edge (a, b) as meaning that a has to come before b—thus an edge defines a precedence relation. A topological order is an order of the vertices that satisfies all the edges.

- Example: Dressing (arrow implies "must come before")



We want to compute order in which to get dressed. One possibility:



The given order is one possible topological order.

2 Does a tpopological order always exist?

- If the graph has a cycle, a topological order cannot exist. Imagine the simplest cycle, consisting of two edges: (a, b) and (b, a). A topological ordering , if it existed, would have to satisfy that a must come before b and b must come before a. This is not possible.
- Now consider we have a graph without cycles; this is usually referred to as a DAG (directed acyclic graph). Does any DAG have a topological order?

The answer is YES. We'll prove this below indirectly by showing that the toposort algorithm always gives a valid ordering when run on any DAG.

3 Topological sort via DFS

It turns out that it's easy to get topological order based on DFS.

Algorithm:

algo #1

1. Call DFS(G) to compute start and finish times for all vertices in G.
2. Return the list of vertices in reverse order of their finish times. That is, the vertex finished last will be first in the topological order, and so on.

Why does this work? We'll start with this claim:

Claim: The vertex finished last by DFS cannot have any incoming edges.

More generally we can prove that:

Claim: Suppose DFS(G) is run on graph G. If G contains an edge (u, v), then the finish time of u is larger than the finish time of v; in other words, u is finished after v. By the Edge Property

Proof: Consider edge (u, v). When this edge is explored, v cannot be GRAY, because in that case v would be an ancestor of u and (u, v) would be a back edge meaning a cycle would exist. So v has to be either BLACK or WHITE. If v is WHITE, it becomes a descendant of u, and it is finished before u is finished. If v is BLACK, it's already been finished so u will be finished after v. Thus for any edge (u, v) we have that u is finished after v is finished.

4 A standalone algorithm for topological sort

Intuitively we see that a vertex that has no incoming edges has to be first. This suggests the following algorithm:

algo #2

- find a vertex with no incoming edges, put it in the output;
- delete all its outgoing edges;
- repeat.

Why does this work?

Claim: A directed acyclic graph always contains a vertex of indegree zero.

Proof: Assume by contradiction that *all* vertices in G have at least one incoming edge. Consider an arbitrary vertex v; it must have an incoming edge, call it (v_1, v); u must have an incoming edge, (v_2, v_1); v_2 must have an incoming edge, call it (v_3, v_2); and so on, we can do this forever; since the graph is finite, this means at some point we must hit the same vertex $\Longrightarrow$ cycle. Contradiction. So it must be that not all vertice have an incoming edge, that is, there must exist at least one vertex with no incoming edges.

Problem: Implement this algorithm in time $O(|V| + |E|)$.

5 Applications