Topologically Sorting a Directed Acyclic Graph

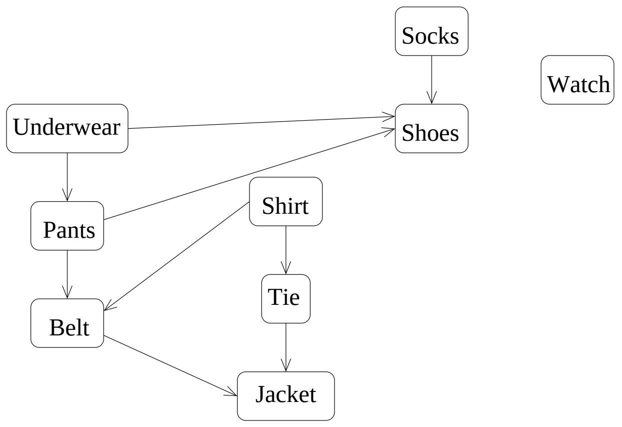
Module 5: Graphs

1 Overview

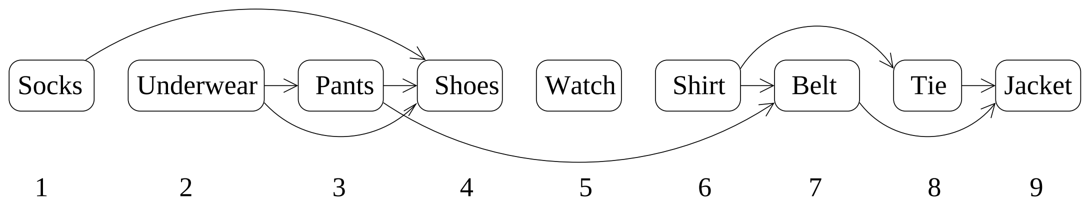
Today we talk about *topological sort*. Topological sort is defined on directed graphs which do not have cycles (called: directed acyclic graphs); it corresponds to ordering the vertices in such a way such that the edges go "forward". If we think of edges as representing precendence, i.e. the start point must come before the endpoint, then a topological sort assures that all the edges are "fullfilled". Topological sort is a very useful concept on directed acyclic graphs. Many problems admit simpler and faster solutions on this class of graphs, and the solutions involve in one way or another the fact that an acyclic graph can be topologically sorted. For example, we'll see that computing shortest paths on directed acyclic graphs can be done in linear time; also the problem of computing longest paths, which is known to be NP-complete on general graphs, can be solved in linear time on directed acyclic graphs.

2 Topological Sort: The problem

- A topological sorting of a *directed acyclic graph* $G = (V, E)$ is a linear ordering of vertices V such that for any edge $(u, v) \in E$, vertex u appears before v in this ordering.
- We think of an edge (u, v) as meaning that u has to come before v—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 (Note: A graph may have many different topological orders; we want to compute one of them).

3 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.

4 Algorithms for computing a topological order

We'll see two algorithm for computing a topological order, both of which run in linear time $O(V+E)$. The first one uses DFS, and teh second one is standalone.

4.1 Topological sort via DFS

```
TopologicalSort-using-DFS(graph G):

  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.
```

Correctness: Why does this work?

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 DAG 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.

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.

Analysis: Computing topological order via DFS runs in $O(V + E)$ time.

4.2 A standalone algorithm for topological sort

Intuitively, in order to sort topologically, we want to start with a vertex that has no incoming edges. This suggests the following algorithm:

```
TopologicalSort(graph G)

    • find a vertex with no incoming edges, put it in the output;

    • delete all its outgoing edges;

    • repeat.
```

Correctness: Clearly, a vertex without incoming edges can be first on topological order. But, is such a vertex guaranteed to exist? What about in the loop?

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.

Analysis: A straightforward implementation of the algorithm above spends $O(V)$ to find a vertex with no incoming edges; this results in $O(V^2)$ total time.

Self-study problem: Implement the algorithm above in time $O(|V| + |E|)$.