Topological Sort

A cycle in a diagraph or directed graph G is a set of edges, $\{(v_1, v_2), (v_2, v_3), ..., (v_{r-1}, v_r)\}$ where $v_1 = v_r$. A diagraph is acyclic if it has no cycles. Such a graph is often referred to as a directed acyclic graph, or DAG, for short. DAGs are used in many applications to indicate precedence among events. For example, applications of DAGs include the following:

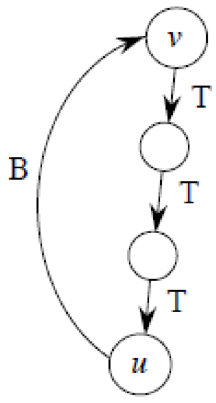
- Inheritance between C++ classes or Java interfaces.
- Prerequisites between courses of a degree program.
- Scheduling constraints between the tasks of a projects.

Thus, a DAG is good for modeling processes and structures that have a **partial order**: $a > b$ and $b > c$ imply $a > c$. But may have a and b such that neither $a > b$ or $b > a$.

One can always make a **total order** (either $a > b$ or $b > a$ for all $a \neq b$) from a partial order. In fact, that's what a **topological sort** will do.

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

Proof First, we argue the necessity (the "only if" part of the statement). That is, show that back edge implies cycle.



Suppose there is a back edge (u, v). Then v is ancestor of u in depth-first forest. Therefore, there is a path from v to u, so v to u to v is a cycle.

Now, we argue the sufficiency (the "if" part of the statement). That is, show that cycle implies back edge. Suppose G contains cycle c. Let v be the first vertex discovered in c, and let (u, v) be the preceding edge in c. At time d[u], vertices of c form a white path from v to u (since v is the first vertex discovered in c). By white-path theorem, u is descendant of v in depth-first forest. Therefore, (u, v) is a back edge. And this completes the proof.

The vertices of a DAG can be ordered in such a way that every edge goes from an earlier vertex to a later vertex. This is called a topological sort or topological ordering. Formally, we say a topological sort of a directed acyclic graph G is an ordering of the vertices of G such that for every edge (v_i, v_j) of G we have $i < j$. That is, a topological sort is a linear ordering of all its vertices such that if DAG G contains an edge (v_i, v_j), then v_i appears before v_j in the ordering. If DAG is cyclic then no linear ordering is possible. In simple words, a topological ordering is an ordering such that any directed path in DAG G traverses vertices in increasing order. It is important to note that if the graph is not acyclic, then no linear ordering is possible. That is, we must not have circularities in the directed graph. For example, in order to get a job you need to have work experience, but in order to get work experience you need to have a job (sounds familiar?)

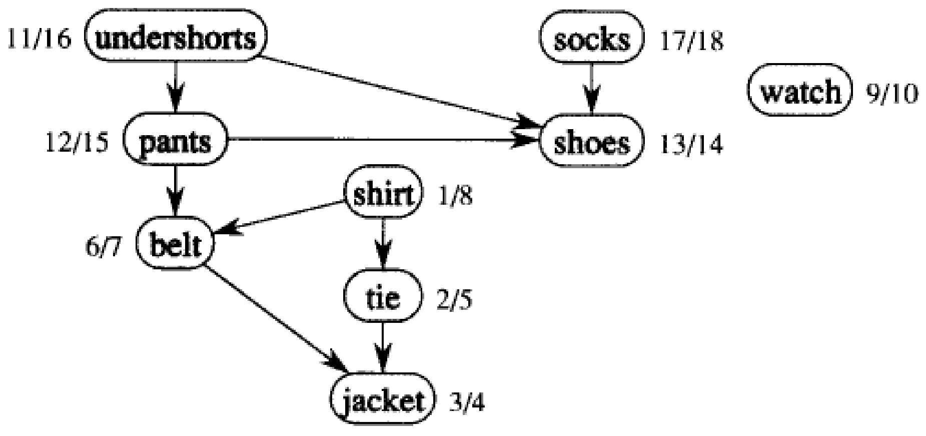
Note that a diagraph may have several different topological sorts. Thus, topological sort is different from the usual kind of "sorting" studied in part 1 of this course.

One way to get a topological sort of a DAG is to run a depth-first search and then order the vertices so their f time-stamps are in descending order. This requires $\Theta(|V| \lg |V|)$ time if a comparison sort algorithm is used.

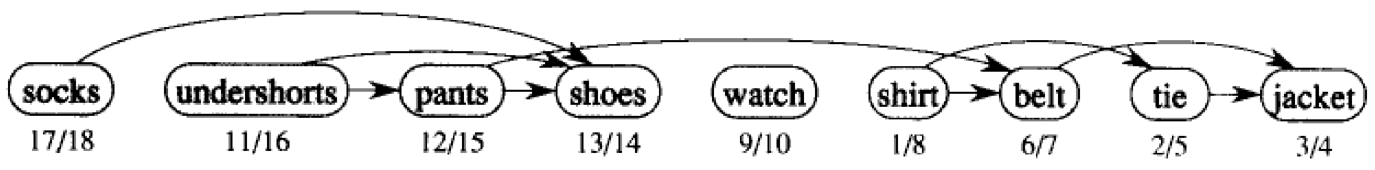
One can modify the depth-first search of a DAG to produce a topological sort. When a vertex changes to black push it on a stack or put it on the front of a linked list. In which case, the running time is $\Theta(|V| + |E|)$.

Example: The following figure (CLRS-Figure 22.7) gives an example that arises when Professor Bumstead gets dressed in the morning. The DAG of dependencies for putting clothing is given (not shown in the figure). (a) The discovery and finishing times from depth-first search are shown next to each vertex. (b) Then the same DAG shown topologically sorted.

(a)



(b)



Topological sort of a DAG: a linear ordering of vertices such that if (u, v) in E, then u appears somewhere before v. (Certainly not like sorting numbers.)

TOPOLOGICAL-SORT(V, E)

```
Call DFS(V, E) to compute finishing times f[v] for all v in V
Output vertices in order of decreasing finish times
```

It should be clear from above discussion that we don't need to sort by finish times.

- We can just output vertices as they are finished and understand that we want the reverse of this list.
- Or we can put vertices onto the front of a linked list as they are finished. When done, the list contains vertices in topologically sorted order.

Example: From the example above.

Order of vertices:

18	socks
16	underpants
15	pants
14	shoes
10	watch
8	shirt
7	belt
5	tie
4	jacket

Correctness

To established the correctness, we just need to show if (*u*, *v*) in E, then f[*v*] < f[*u*]. When we explore (*u*, *v*), what are the colors of *u* and *v*?

The vertex *u* is gray. Now the first question is: Is *v* gray, too? The answer is "no", because then *v* would be ancestor of *u*. Which implies (*u*, *v*) is a back edge. But this contradicts the previous lemma (DAG has no back edges). The second question is: Is *v* white? If *v* is white then *v* becomes descendant of *u*. By parenthesis theorem, d[*u*] < d[*v*] < f[*v*] < f[*u*]. The third question is: Is *v* black? If *v* is black, then *v* is already finished. Since, we are exploring (*u*, *v*), we have not yet finished *u*. Therefore, f[*v*] < f[*u*].

Back

Dated: February 9, 2010.