DAGS and Topological Sort

A graph is *acyclic* if does not contain a cycle. A *d*irected *a*cyclic *g*raph is called a *DAG*.

To the right is an example of a DAG. The nodes are some of the courses that can be taken to satisfy a computer science (or other) major at Cornell. A directed edge from CS course c1 to CS course c2 is drawn if c1 is a prerequisite for c2. In the full graph for the CS major, there better not be a cycle! One can use the graph to find the longest prerequisite chain —it shouldn’t be more  
than 8 course, and preferably less, so students can graduate in eight semesters.

CS1110

CS2110

CS2800

CS3110

DAGs can be used to represent scheduling problems, where certain tasks must be done before others. The tasks are the nodes of a graph, and a directed edge is drawn to indicate that one task (the source of the edge) must be done before the second task (the sink of the edge). Laying out tasks to be done on a factory floor and figuring out the order in which to put clothes on (socks before shoes!) are examples.

Website <https://www.genealogy.math.ndsu.nodak.edu> contains the genealogy tree of mathematicians and computer scientists, with over 230,675 entries as of July 1918. Draw an arrow from each PhD to that PhD’s advisors (a PhD has one or two advisors). The result is a DAG and not a tree. An example that shows why is found in David Gries’s tree, which is found here: In this tree, von Vega advised von Gruber and von Maffei, and both of them advised von Neuhaus.

von Vega

von Gruber

von Maffei

von Neuhaus

**Topological sort**

Suppose we want to order the nodes of a DAG so that the source of each directed edge comes before its sink. With the prerequisite graph shown above, there are two possible orderings:

CS1110, CS2110, CS2800, CS3110  
 CS1110, CS2800, CS2110, CS3110

Ordering the nodes of a DAG in this fashion is called *topological sort*. We develop an algorithm for performing a topological sort, giving first an abstract algorithm based on a neat idea and then figuring out how to implement it efficiently.

**The basic idea**

A DAG must have a node with indegree 0 —if not, there would be a cycle and it would not be a DAG. So, choose a node with indegree 0, make it the first node in the ordering, and delete it and all edges leaving it from the graph.

For the prerequisite graph shown above, CS1110 has indegree 0. Therefore, it comes first, and the graph is changed to the one on the right.

CS2110

CS2800

CS3110

Repeat the process: Choose a node with indegree 0 —there are two, CS2110 and CS2800, and either one can be chosen— make it the next node in the ordering; and delete it and all edges leaving it from the graph.

Thus, we have this little algorithm:

**while** (the graph has a node) {  
 Choose a node w (say) with indegree 0;  
 Make w the next node in the ordering;  
 Delete w and all edges leaving it from the graph  
 }

Note also that if at some point the graph contains a node but no node has indegree 0, then there is a cycle. Thus, this little algorithm can be modified to determine whether a digraph is indeed a DAG.

**Writing an efficient Java program for topological sort**

The algorithm written above is not realistic. We shouldn’t be allowed to change the graph. So we