Topological Sort

See pages 647 to 650 of the text.

Theorem: In a directed graph, there must either be a cycle or a a node with no incoming edges.
Proof: Start with any node in the graph; call it x_1. If this node has no incoming edges we are done. If it does have an incoming edge, call the source of this edge x_2. So we have

$$x_2 \text{ -> } x_1$$

Now consider x_2. If it has no incoming edges we are done. If it does have an incoming edge, call the source of this edge x_3. So now we have

$$x_3 \text{ -> } x_2 \text{ -> } x_1$$

Continuing in this way we either find a node with no incoming edge or a path of length n:

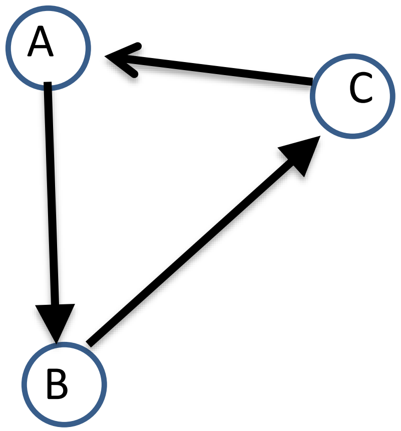
$$x_{n+1} \text{ -> } x_n \text{ -> } x_{n-1} \text{ -> } .... \text{ -> } x_2 \text{ -> } x_1$$

There are n+1 nodes on this path. If n is the number of nodes in the entire graph, some node must appear more than once on this path, and that gives a cycle.

Moral: In any DAG there is a node with no incoming edges.

A *topological sort* of a graph is an ordering of the nodes of the graph that is consistent with the edges of the graph: if there is an edge from node x to node y then x comes before y in the ordering. This is a commonly used technique for scheduling problems, where there are constraints among the items to be scheduled. It is also used to update cells in spreadsheets, where a change to any one cell could lead to the modification of many other cells in the sheet.

It should be clear that if a graph contains a cycle then it has no topological sort, for there is no ordering of the cycle nodes that is consistent with the edges of the graph.



However, for an acyclic directed graph there is an easy algorithm for finding a topological sort:

Topological Sort Algorithm: Maintain a WorkingSet of graph nodes. Initialize the WorkingSet to contain any node of the graph that has no incoming edges.

Repeat the following steps until the WorkingSet is empty.

a) Remove any one node from the WorkingSet. Call this node X.
b) Remove every edge from node X to any other node Y.
c) If node Y has no other incoming edges, add node Y to the WorkingSet.

Continue these steps until the WorkingSet is empty.

How do we know this works? Note that a node cannot be added to the WorkingSet until all of is incoming edges have been removed, and that does not happen until every node that has an edge to it is added to the output. So the ordering is correct. Also, note that if the graph starts out without a cycle, the steps of this algorithm can't create a cycle. So the nodes and edges that remain form an acyclic graph. According to our theorem, this graph must have a node with no incoming edges, which is removed from the graph and added to the working set. So this algorithm doesn't terminate until all nodes from the graph have been added to the WorkingSet and then added to the output. The algorithm thus outputs all nodes of the graph in a correct order.

How long does this take? If we use a queue to represent the WorkingSet, we can get constant-time inserts and removals. The algorithm visits each node in the graph and removes each edge, so it runs in time proportional to the number of edges: O(|E|).

Clicker Q: Do you see a way to use this to determine *if* a directed graph has a cycle?

A. Yes
B. No