Topological Sorting

The problem of topological sorting is one of ordering (numbering) the nodes in a directed acyclic graph (DAG) in such a way that if there is a nonempty path from node $A$ to node $B$, then $A$ appears before $B$ in that ordering.

Here is an algorithm at the highest level of abstraction.

while not done
    if the graph is empty, then the task is done
        (exit the loop);
    pick a node with no predecessor;
    if no such node exists, then the graph is cyclic
        (exit the loop);
    output that node (number that node);
    delete that node from the graph;
end while

A detailed algorithm follows. It has been adapted from the following book:


Let $G = (V, E)$ be a DAG. Without loss of generality, let $V = \{1, \ldots, n\}$.

/*
 * Output the nodes in the current directed acyclic graph
 * so that if there is a directed path from node A to node
 * B, then A is output before B.
 * Pre: The graph is acyclic
 * Post: The graph may be modified
 * Post: All nodes in the graph are output
 * Post: The output order follows the dependencies of the
 * nodes in the digraph.
 */
Data Structures: The algorithm employs a sequential table $X[1], \ldots, X[n]$ for each node where a node $X[k]$ has the representation

<table>
<thead>
<tr>
<th>COUNT[k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOP[k]</td>
</tr>
</tbody>
</table>

where COUNT[k] is the number of direct predecessors of node $k$, and TOP[k] is a pointer to (the beginning of) the list of direct successors of $k$; an entry on the list is of the form

| SUCC |
| NEXT |

where SUCC is a direct successor of $k$ and NEXT is a pointer to the next item on the list. Further, a queue is maintained to hold nodes $k$ with COUNT[k] = 0.

Note: The graph is dynamically modified during execution of the algorithm. Further, COUNT[k] = 0 if and only if node $k$ has no direct predecessor at that point.

Objective is to systematically output the nodes whose COUNT field is zero. Efficiency is achieved by carefully avoiding the “searching” for nodes whose COUNT field is zero, and this can be done by maintaining a queue containing such nodes.

Algorithm:

1. input the value of $n$, and set $N = n$;
2. for $i \leftarrow 1$ to $n$
   \{ initialize COUNT[i] to zero;
   initialize TOP[i] to nil;
   \}
3. // Input the arcs
   while (list of arcs has not been exhausted)
   \{ input an arc $(j, k)$;
   increment COUNT[k];
   create a node pointed to by (say) $p$;
   set its SUCC field to $k$ and add it to the list having head pointer TOP[j];
   \}
4. // Initialize the queue of the output
   initialize the queue by one scan of the COUNT vector;
   all nodes $j$ with COUNT[j] = 0 are included in the queue;
5. perform the following operations until the queue is empty
   a. remove a node, say $j$, from the queue and output $j$;
   b. for each arc $(i, k)$, decrement COUNT[k] and if this operation causes
      COUNT[k] to become 0, then insert $k$ in the queue;