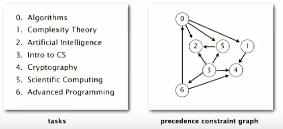
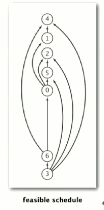
Topological Sort

Goal: Given a set of tasks to be completed with precedence constraints, in which order should we schedule the tasks.

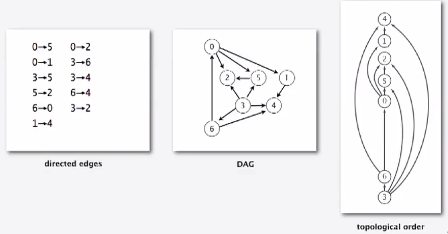
Digraph model: vertex = task; edge = precedence constraint.



End result:



Topological sort works on DAG: directed acyclic graph (directed graph with no cycles)

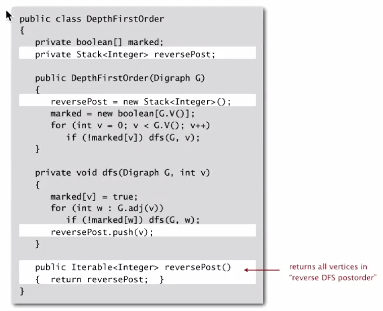


Redraw DAG so that all edges point upwards

Solution: use DFS.

1. Run depth-first search
2. Return vertices in reverse postorder

DFS Top sort implementation

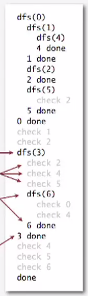


Proposition: Reverse DFS postorder of a DAG is a topological order

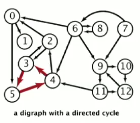
Proof: Consider any edge v->w. When dfs(v) is called:

* **Case 1**: DFS(w) has already been called and returned. Thus, w was done before v.
* **Case 2:** DFS(w) has not yet been called. DFS(w) will get called directly or indirectly by DFS(v) and will finish before DFS(v). Thus w will be done before v.
* **Case 3:** DFS(w) has already been called but has not yet returned. Can’t happen in a DAG; function call stack contains a path from w to v, so v->w would complete a cycle.

Call stack for above:



Directed cycle detection



Proposition: A digraph has a topological order iff no directed cycle

Proof:

* If directed cycle, topological order is impossible
* If no directed cycle, DFS-based algorithm finds a topological order

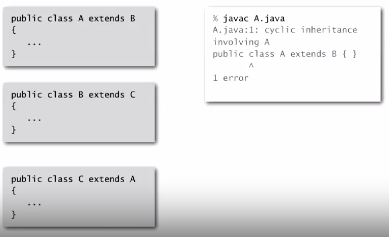
Goal: given a digraph, find a directed cycle  
Solution: DFS.

Directed cycle detection application

Scheduling: Given a set of tasks to be completed with precedence constraints, in what order should we schedule the tasks

Remark: a directed cycle implies scheduling problem is infeasible

Java compiler does cycle detection



Excel does cycle detection (and has circular reference toolbar)

