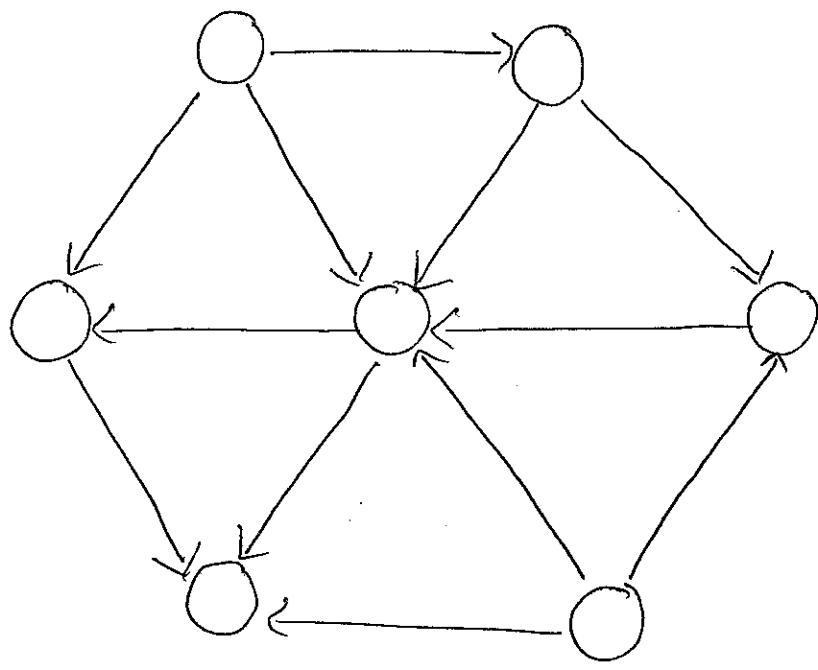


Topological Sorting of Directed Acyclic Graphs.

A directed graph that has no cycles is called a Directed Acyclic Graph.

The ~~nodes~~ vertices of such a graph could represent tasks. An edge from task u to task v means u must complete before v can begin. In such a situation, the requirement that there be no cycles in the graph ~~is~~ is a natural one.



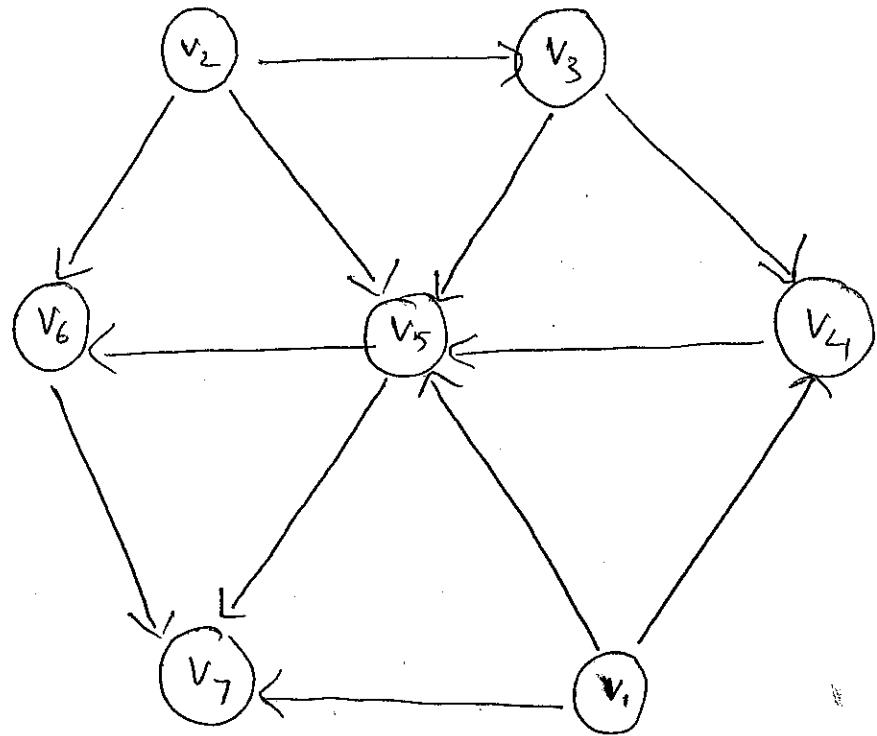
Given such a graph representing tasks and dependencies, it is natural to want to order the tasks so as to respect the precedence relationships.

Problem: Given a directed acyclic graph $G = (V, E)$, find an ordering of the vertices $\{v_1, v_2, \dots, v_n\}$ so

that for any edge $(v_i, v_j) \in E$, we have $i < j$.

Such an ordering is called a topological sort of the vertices.

In the example:



It is easy to see that only graphs with no cycles admit a topological sort.

Claim If G admits a topological sort, G is a DAG.

Now, our algorithm will show as a by product that every DAG has a topological ordering.

The key is the following observation.

Observation: Every DAG has a vertex with no incoming edges.

Proof: In class. See book otherwise.

This suggests the following algorithm to topologically sort DAG G .

- Find a node v with no incoming edges.
- Print v
- Delete v from G along with incident edges.
- Recursively compute a topological sort of remaining graph.

Note that remaining graph is a DAG as well. A proof by induction on the no. of vertices in the graph will show that this algorithm works correctly.