Dynamic Programming

1. Search optimal solution for optimal substructure.
2. Set up a recurrence for the optimal solution.
3. Compute the recurrence bottom-up.

	D & C	vs	DP
Subproblems:	- few - disjoint - much smaller - original problem		- many - overlapping - slightly smaller - parameterized
Resulting Algorithm is:	recursive		iterative

~~Recap~~: Graphs

Graph $G = (V, E)$

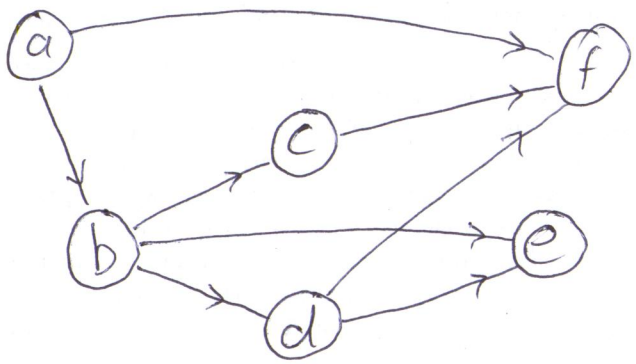
- V is a set of vertices (representing things)
- E is a set of edges (representing connections between things)
- Each edge $e \in E$ is an ordered or unordered pair of distinct vertices from V.
- All edges ordered $\Rightarrow$ G is directed
- All edges unordered $\Rightarrow$ G is undirected

Topological Sort

Input: A directed acyclic graph (DAG) G.

Output: A list L of G's vertices ordered such that if there is a path from u to v in G then u comes before v in L.

Example:

$G =$ 

$L = [a, b, d, e, c, f]$

Algorithm TopoSort(G):

1. for each vertex v: $in(v) \leftarrow 0$
2. for each edge $(u,v) \in E$: $in(v) \leftarrow in(v) + 1$
3. Initialize an empty queue Q and list L
4. for each vertex v: if $in(v) = 0$ then add v to Q
5. while Q is not empty do
6. dequeue v from Q; add v to L
7. for each edge (v,w) do — incident on v
8. $in(w) \leftarrow in(w) - 1$
9. if $in(w) = 0$ then add w to Q
10. return L

- Correct? Loop Invariant:

Before each and after iteration of the while loop, in(v) corresponds to the in-degree of v in G-L, and Q contains all vertices with in-degree 0 in G-L.

Lemma 1: Every DAG has a vertex with in-degree 0.

Proof: Start at any vertex and follow edges backwards. This process cannot repeat vertices, otherwise there is a cycle. Thus, it must stop after at most n-1 steps at a vertex with no incoming edges. □

Lemma 2: Removing a vertex from a DAG results in a DAG.

Proof: The new graph is still directed. Any cycle in the new graph must already have been present in the old graph, since no new edges are introduced, but there are no cycles in the old graph. □

Therefore, the while loop only finishes when L contains all vertices in G. And since a vertex $v \in Q$ has in-degree 0 in G-L, there is no path from any remaining vertex to v, so v can be added to L without violating the criterium.

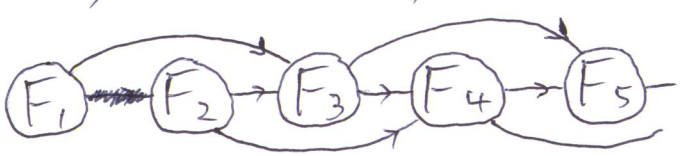
- Terminate?

Vertices are added to Q only once: when they first get in-degree 0. Every iteration removes an element from Q, so it will become empty at some point.

- Efficient?

Line	Work		Executions		Total
1	$O(\|V\|)$	×	1	=	$O(\|V\|)$
2	$O(\|E\|)$	×	1	=	$O(\|E\|)$
3	$O(1)$	×	1	=	$O(1)$
4	$O(\|V\|)$	×	1	=	$O(\|V\|)$
5	$O(1)$	×	$\|V\|$	=	$O(\|V\|)$
6	$O(1)$	×	$\|V\|$	=	$O(\|V\|)$
7	$O(1)$	×	$\|E\|$	=	$O(\|E\|)$
8	$O(1)$	×	$\|E\|$	=	$O(\|E\|)$
9	$O(1)$	×	$\|E\|$	=	$O(\|E\|)$
10	$O(1)$	×	1	=	$O(1)$ +
					$O(\|V\|+\|E\|)$

The subproblems in a DP algorithm always form a DAG, with edges indicating dependency.

Example: 

The best order to solve these in, is a topological sort of this graph!