3.6 DAGs and Topological Ordering
Connectivity in Directed Graphs

Q. How to determine if G is strongly connected, in $O(m + n)$ time?
Connectivity in Directed Graphs

Q. How to determine if G is strongly connected, in $O(m + n)$ time?

A. BFS($G, s$) and BFS($G^{rev}, s$)
Q. Does this directed graph contain a cycle?

Q. How to define a directed cycle?
Connectivity in Directed Graphs

Def. A **path** in a *directed* graph $G = (V, E)$ is a sequence $P$ of nodes $v_1, v_2, \ldots, v_k$ with the property that each consecutive pair $v_i, v_{i+1}$ is joined by a *directed* edge $(v_i, v_{i+1})$ in $E$.

Def. A **directed cycle** in a directed graph $G$ is a path $v_1, v_2, \ldots, v_k$ in $G$ in which $v_1 = v_k$, $k > 2$, and the first $k-1$ nodes are all distinct.
Directed Acyclic Graphs

**Def.** A DAG is a directed graph that contains no directed cycles.

**Ex.** Precedence constraints: edge \((v_i, v_j)\) means \(v_i\) must precede \(v_j\).
Often used in **planning** algorithms.

![A DAG](image)
Q. Is a tree with directed edges a DAG?

Q. Is a DAG a tree?
Def. A topological ordering of a directed graph $G = (V, E)$ is an ordering of its nodes as $v_1, v_2, \ldots, v_n$ so that for every edge $(v_i, v_j)$ we have $i < j$.

[all arrows point to the right]

Ex. Check whether a set of precedence relations between (birth and death) events is chronologically consistent.

\begin{figure}
\centering
\begin{subfigure}{0.45\textwidth}
\centering
\begin{tikzpicture}
  \node (v1) at (0,0) [shape=circle,draw,fill=black!10] {$v_1$};
  \node (v2) at (1,1) [shape=circle,draw,fill=black!10] {$v_2$};
  \node (v3) at (2,0) [shape=circle,draw,fill=black!10] {$v_3$};
  \node (v4) at (1,-1) [shape=circle,draw,fill=black!10] {$v_4$};
  \node (v5) at (0,-2) [shape=circle,draw,fill=black!10] {$v_5$};
  \node (v6) at (-1,-1) [shape=circle,draw,fill=black!10] {$v_6$};
  \node (v7) at (-2,0) [shape=circle,draw,fill=black!10] {$v_7$};

  \draw [->] (v1) -- (v2);
  \draw [->] (v2) -- (v3);
  \draw [->] (v3) -- (v4);
  \draw [->] (v4) -- (v5);
  \draw [->] (v5) -- (v6);
  \draw [->] (v6) -- (v7);
  \draw [->] (v7) -- (v1);
\end{tikzpicture}
\caption{a DAG}
\end{subfigure}
\hfill
\begin{subfigure}{0.45\textwidth}
\centering
\begin{tikzpicture}
  \node (v1) at (0,0) [shape=circle,draw,fill=black!10] {$v_1$};
  \node (v2) at (1,1) [shape=circle,draw,fill=black!10] {$v_2$};
  \node (v3) at (2,0) [shape=circle,draw,fill=black!10] {$v_3$};
  \node (v4) at (1,-1) [shape=circle,draw,fill=black!10] {$v_4$};
  \node (v5) at (0,-2) [shape=circle,draw,fill=black!10] {$v_5$};
  \node (v6) at (-1,-1) [shape=circle,draw,fill=black!10] {$v_6$};
  \node (v7) at (-2,0) [shape=circle,draw,fill=black!10] {$v_7$};

  \draw [->] (v1) -- (v2);
  \draw [->] (v2) -- (v3);
  \draw [->] (v3) -- (v4);
  \draw [->] (v4) -- (v5);
  \draw [->] (v5) -- (v6);
  \draw [->] (v6) -- (v7);
  \draw [->] (v7) -- (v1);
\end{tikzpicture}
\caption{a topological ordering}
\end{subfigure}
\end{figure}
Precedence Constraints

Precedence constraints. Edge \((v_i, v_j)\) means task \(v_i\) must occur before \(v_j\).

Applications.
- Course prerequisite graph: course \(v_i\) must be taken before \(v_j\).
- Compilation: module \(v_i\) must be compiled before \(v_j\).
- Pipeline of computing jobs: output of job \(v_i\) needed to determine input of job \(v_j\).
- Software development planning: some modules must be written before others (with durations \(\rightarrow\) critical path)

Q. How can we find out whether a topological order exists?
Lemma 3.18. If G has a topological order, then G is a DAG.

Pf.

Q. How to prove an “if ..., then...”?
Lemma 3.18. If $G$ has a topological order, then $G$ is a DAG.

Pf. Suppose that $G$ has a topological order $v_1, \ldots, v_n$.

Q. What proof technique to use now?

... 

So $G$ is a DAG. □

the supposed topological order: $v_1, \ldots, v_n$
Lemma 3.18. If G has a topological order, then G is a DAG.

Pf. (by contradiction)

Suppose that G has a topological order $v_1, \ldots, v_n$.
Suppose that G has a directed cycle C.

Q. How to derive a contradiction?

\[ \begin{align*}
&\text{Contradiction.} \\
&\text{So G has no cycle. So G is a DAG.} \quad \blacksquare
\end{align*} \]
Lemma 3.18. If G has a topological order, then G is a DAG.

Pf. (by contradiction)
Suppose that G has a topological order \( v_1, \ldots, v_n \).
Suppose that G has a directed cycle C.
Let \( v_i \) be the lowest-indexed node in C, and let \( v_j \) be the node just before \( v_i \) in C; thus \( (v_j, v_i) \) is an edge.
By our choice of i, we have \( i < j \).
On the other hand, since \( (v_j, v_i) \) is an edge and \( v_1, \ldots, v_n \) is a topological order, we must have \( j < i \), a contradiction.
So G has no cycle. So G is a DAG. •

This proof can be found on page 101.
Directed Acyclic Graphs

Lemma 3.18. If G has a topological order, then G is a DAG.

Q. Does every DAG have a topological ordering?
Q. If so, how do we compute one?
Directed Acyclic Graphs

Lemma 3.18. If $G$ has a topological order, then $G$ is a DAG.

Q. Does every DAG have a topological ordering?
Q. If so, how do we compute one?

Idea: start with node with no incoming edge
Q. Does such a node always exist in a DAG?
Lemma 3.19. If $G$ is a DAG, then $G$ has a node with no incoming edges.

Pf.

Q. How to prove an “if ..., then...”? 
Lemma 3.19. If G is a DAG, then G has a node with no incoming edges.

Pf.

Suppose that G is a DAG.

Q. What proof technique to use now?

... 

So there must be a node with no incoming edges. □
**Lemma 3.19.** If G is a DAG, then G has a node with no incoming edges.

**Pf.** (by contradiction)
Suppose that G is a DAG.
Suppose every node has at least one incoming edge.

**Q.** How to derive a contradiction?

... 

Contradiction.
So there must be a node with no incoming edges.

\[ \Box \]
Lemma 3.19. If $G$ is a DAG, then $G$ has a node with no incoming edges.

Pf. (by contradiction)
Suppose that $G$ is a DAG.
Suppose every node has at least one incoming edge.
Pick any node $v$, and follow edges backward from $v$.
Repeat until we visit a node, say $w$, twice.
Let $C$ denote the sequence of nodes encountered between successive visits to $w$. $C$ is a cycle.
Contradiction.
So there must be a node with no incoming edges. □

This proof can be found on page 102.
Directed Acyclic Graphs

Lemma 3.20. If G is a DAG, then G has a topological ordering.

Pf.
Idea of proof: add nodes without incoming edge one by one to topological ordering

Q. What proof technique can reflect this iterative procedure?
Lemma 3.20. If $G$ is a DAG, then $G$ has a topological ordering.

Pf. (by induction on $n$)
Lemma 3.20. If $G$ is a DAG, then $G$ has a topological ordering.

Pf. (by induction on $n$)

Base case:
Hypothesis:
Step:
Lemma 3.20. If G is a DAG, then G has a topological ordering.

Pf. (by induction on n)

Base case: true if n = 1, because topological ordering is G.

Hypothesis: If G is DAG of size $\leq n$, then G has a topological ordering.

Step:

Q. What to prove here?
Lemma 3.20. If G is a DAG, then G has a topological ordering.

Pf. (by induction on n)

Base case: true if n = 1, because topological ordering is G.

Hypothesis: If G is DAG of size \( \leq n \), then G has a topological ordering.

Step: Given DAG G’ with n+1 nodes.

... using inductive hypothesis (IH)

Create topological ordering for G:

- ...

By induction the lemma is proven. \( \square \)
Lemma 3.20. If $G$ is a DAG, then $G$ has a topological ordering.

Pf. (by induction on $n$)

Base case: true if $n = 1$, because topological ordering is $G$.

Hypothesis: If $G'$ is DAG of size $\leq n$, then $G'$ has topological ordering.

Step: Given DAG $G$ with $n+1$ nodes.

Find a node $v$ with no incoming edges.

$G - \{v\}$ is a DAG, since deleting $v$ cannot create cycles.

By inductive hypothesis (IH), $G - \{v\}$ has a topological ordering.

Create topological ordering for $G$:

- Place $v$ first; then append topological ordering of $G - \{v\}$.
- This is valid since $v$ has no incoming edges.

By induction the lemma is proven. □

This proof can be found on page 102.
Q. Give an algorithm to return a topological ordering.

To compute a topological ordering of $G$:
Find a node $v$ with no incoming edges and order it first
Delete $v$ from $G$
Recursively compute a topological ordering of $G-\{v\}$
and append this order after $v$
Q. Give a topological sort of the following graph.
Theorem. Algorithm finds a topological order in $O(m + n)$ time.

Q. How to implement? Which information do you need to maintain?
Theorem. Algorithm finds a topological order in $O(m + n)$ time.

Pf.

Maintain the following information:
- $\text{count}[w] = \text{remaining number of incoming edges in } w$
- $S = \text{set of remaining nodes with no incoming edges}$

Initialization: $O(m + n)$ via single scan through graph.

Update: to delete $v$
- remove $v$ from $S$
- decrement $\text{count}[w]$ for all edges from $v$ to $w$, and add $w$ to $S$ if $\text{count}[w]$ hits $0$
- this is $O(1)$ per edge