Describe an algorithm to determine if an undirected graph has a cycle

Cycle finding
- Does a graph have a cycle?
- Find a cycle
- Find a cycle through a specific vertex v
  - Linear runtime: $O(n+m)$

Find a cycle through a vertex v
- Not obvious how to do this with BFS from vertex v

Depth First Search
- Each edge goes between vertices on the same branch
- No cross edges
A DFS from vertex v gives a simple algorithm for finding a cycle containing v. How does this algorithm work and why?

Connected Components
- Undirected Graphs

Directed Graphs
- A Strongly Connected Component is a subset of the vertices with paths between every pair of vertices.

Identify the Strongly Connected Components

Topological Sort
- Given a set of tasks with precedence constraints, find a linear order of the tasks

Find a topological order for the following graph
If a graph has a cycle, there is no topological sort

- Consider the first vertex on the cycle in the topological sort
- It must have an incoming edge

Lemma: If a graph is acyclic, it has a vertex with in degree 0

- Proof:
  - Pick a vertex $v_1$, if it has in-degree 0 then done
  - If not, let $(v_2, v_1)$ be an edge, if $v_2$ has in-degree 0 then done
  - If not, let $(v_3, v_2)$ be an edge . . .
  - If this process continues for more than $n$ steps, we have a repeated vertex, so we have a cycle

Topological Sort Algorithm

While there exists a vertex $v$ with in-degree 0

Output vertex $v$

Delete the vertex $v$ and all outgoing edges

Details for $O(n+m)$ implementation

- Maintain a list of vertices of in-degree 0
- Each vertex keeps track of its in-degree
- Update in-degrees and list when edges are removed
- $m$ edge removals at $O(1)$ cost each