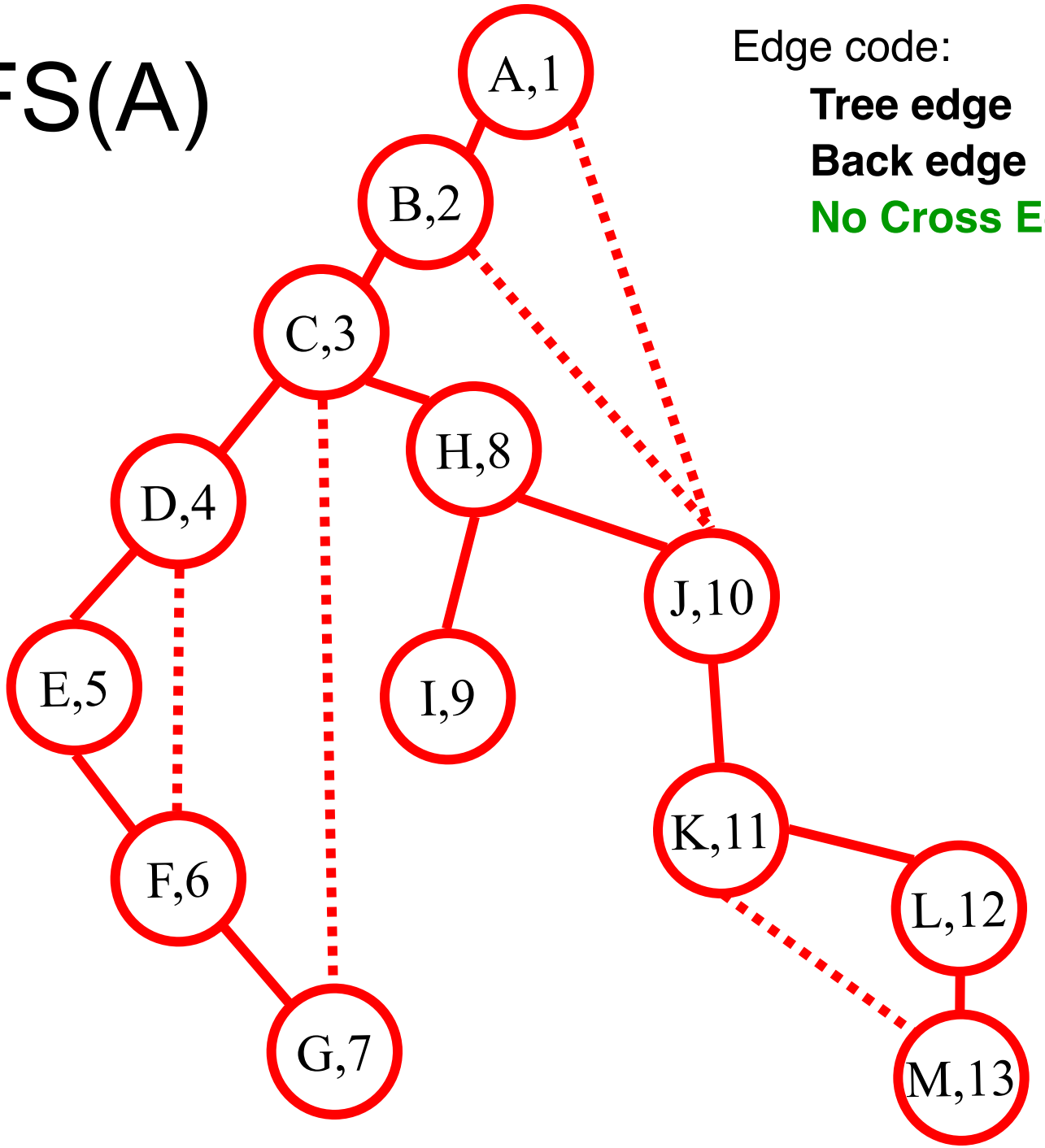
CS 401: Computer Algorithm I

DFS / Topological Ordering

Xiaorui Sun

DFS(A)



Edge code:

Tree edge

Back edge

No Cross Edges!

Properties of (undirected) DFS

Like BFS(s):

- DFS(s) visits x iff there is a path in G from s to x

  So, we can use DFS to find connected components
- Edges into then-undiscovered vertices define a ***tree*** – the "DFS tree" of G

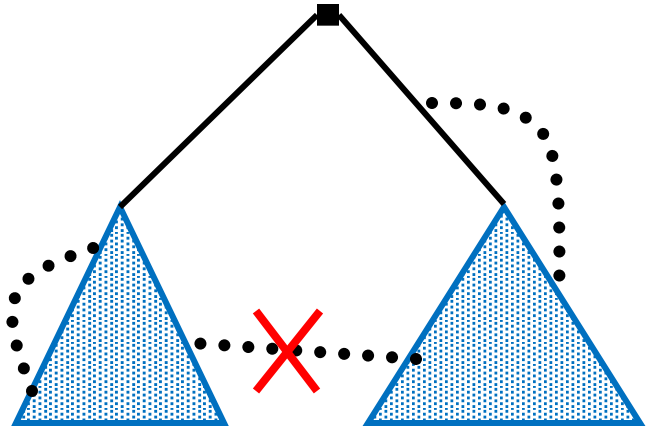
Unlike the BFS tree:

- The DFS tree isn't minimum depth
- Its levels don't reflect min distance from the root
- Non-tree edges never join vertices on the same or adjacent levels

Non-Tree Edges in DFS

BFS tree ≠ DFS tree, but, as with BFS, DFS has found a tree in the graph s.t. non-tree edges are "simple" in some way.

All non-tree edges join a vertex and one of its descendents/ancestors in the DFS tree



This property is useful for an O(n+m) time algorithm of Homework 1 Problem 5

Non-Tree Edges in DFS

Lemma: For every edge $\{x, y\}$, if $\{x, y\}$ is not in DFS tree, then one of x or y is an ancestor of the other in the tree.

Proof:

Suppose that x is visited first.

Therefore DFS(x) was called before DFS(y)

Since $\{x, y\}$ is not in DFS tree, y was visited when the edge $\{x, y\}$ was examined during DFS(x)

Therefore y was visited between the start of DFS(x) and the examination of $\{x, y\}$ in DFS(x)

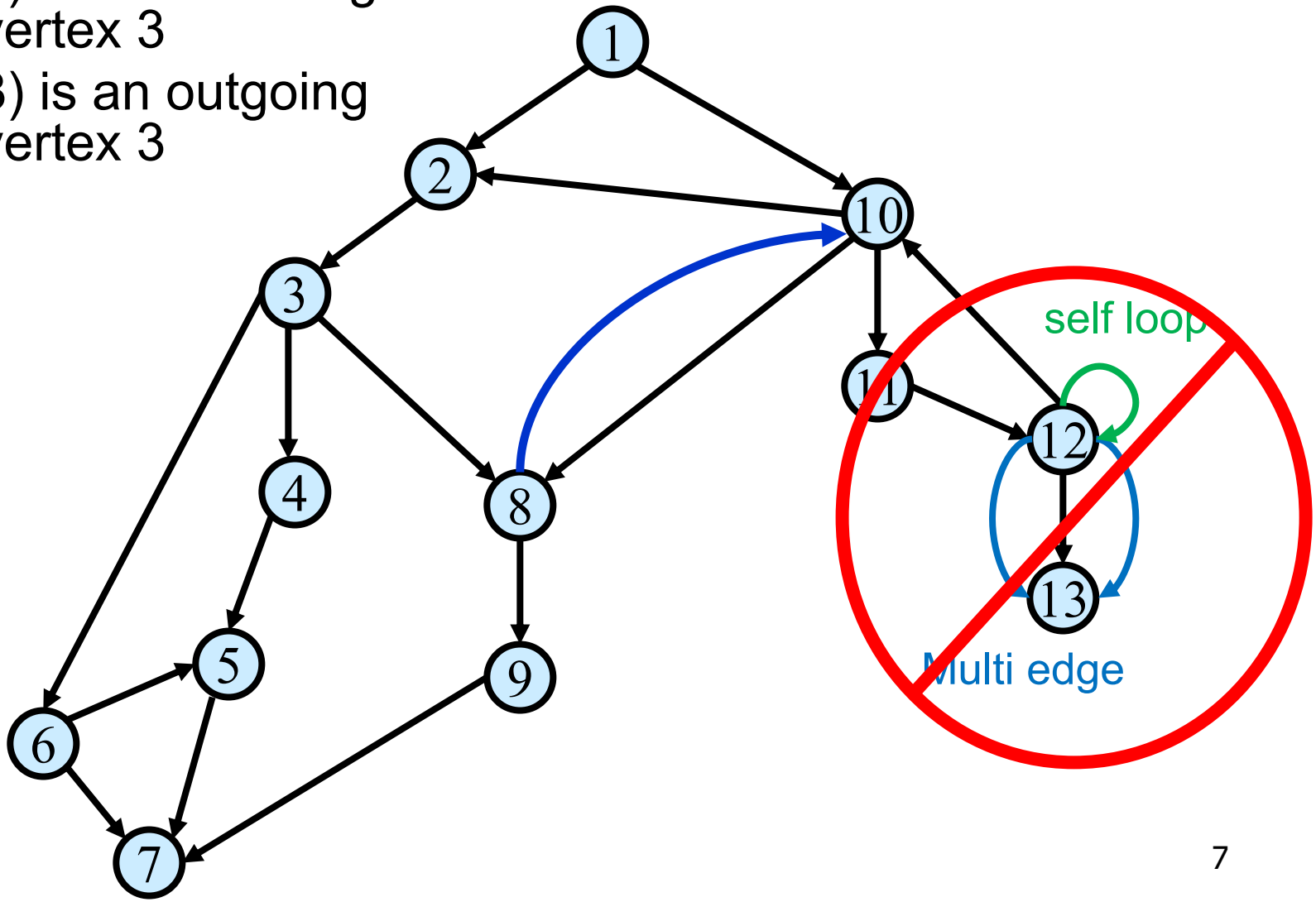
So y is a descendant of x.

Directed Graph and Topological Ordering

Directed Graphs

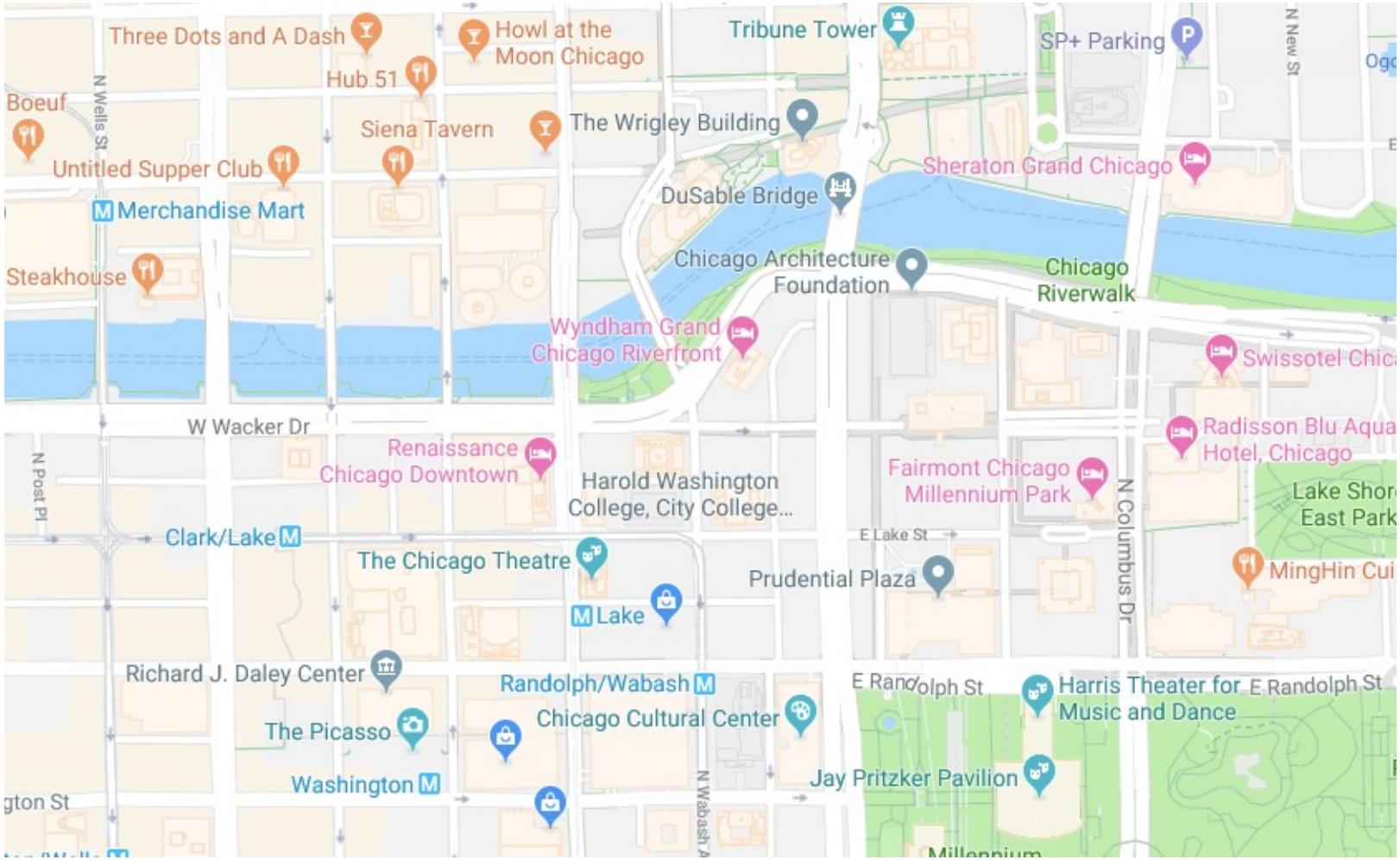
edge (2, 3) is an incoming edge for vertex 3

edge (3, 8) is an outgoing edge for vertex 3



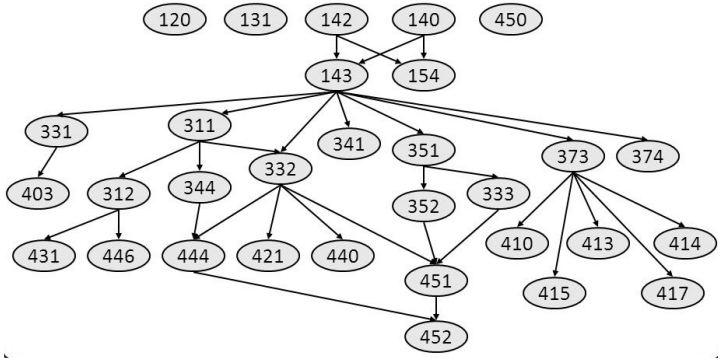
Directed Graphs

Node = intersection, edge = one-way street



Precedence Constraints

In a directed graph, an edge (i, j) means task i must occur before task j.



Applications

- Course prerequisite:

  course i must be taken before j

- Compilation:

  must compile module i before j

- Computing overflow:

  output of job i is part of input to job j

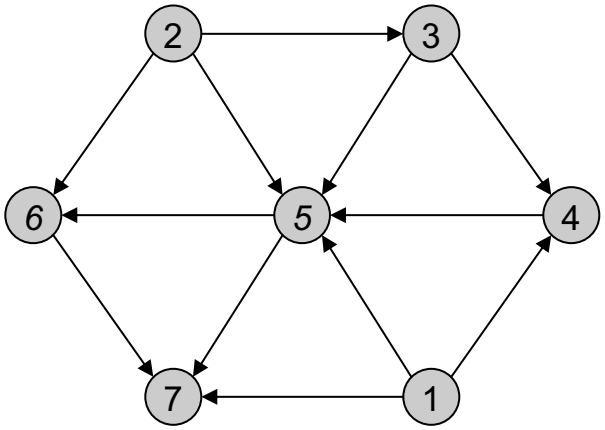
- Manufacturing or assembly:

  sand it before paint it

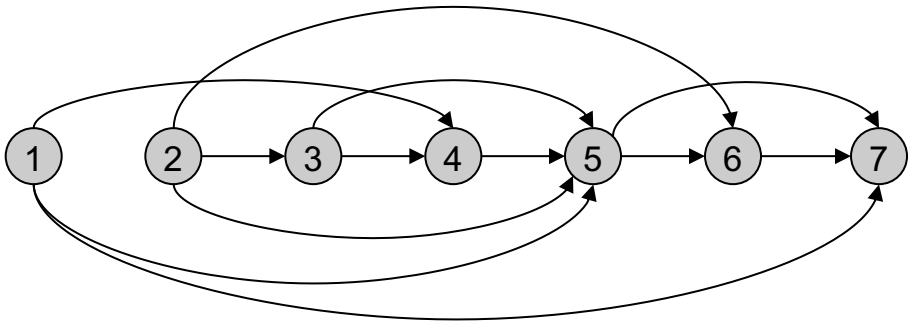
Directed Acyclic Graphs (DAG)

Def: A DAG is a directed acyclic graph, i.e., one that contains no directed cycles.

Def: A topological order 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 $i < j$.



a DAG



a topological ordering of that DAG– all edges left-to-right

DAGs: A Sufficient Condition

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

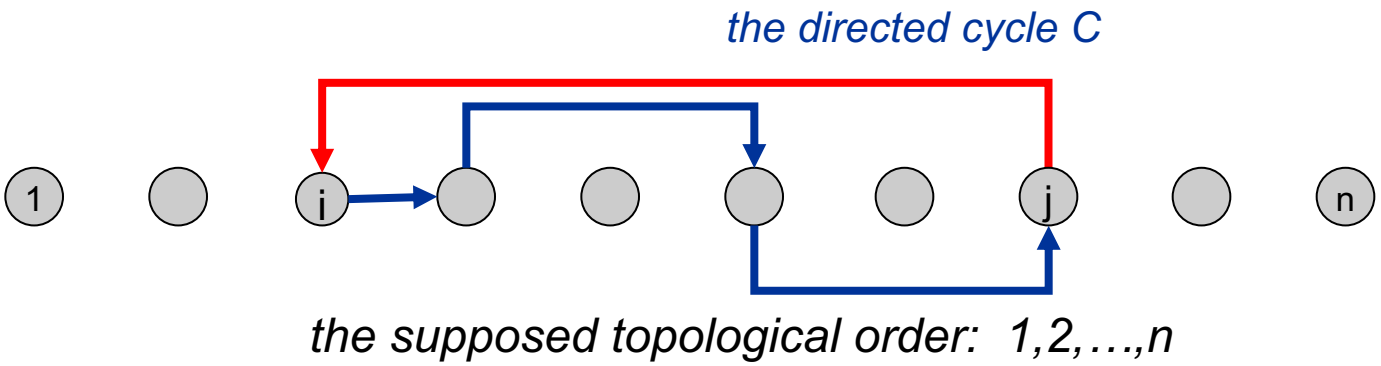
Proof. (by contradiction)

Suppose that G has a topological order $1,2,\ldots,n$ and that G also has a directed cycle C.

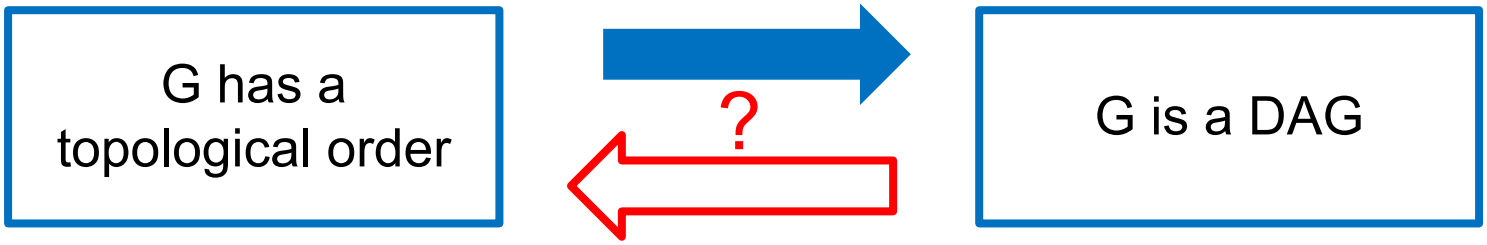
Let i be the lowest-indexed node in C, and let j be the node just before i; thus (j,i) is an (directed) edge.

By our choice of i, we have $i < j$.

On the other hand, since (j,i) is an edge and $1,\ldots,n$ is a topological order, we must have $j < i$, a contradiction



DAGs: A Sufficient Condition



Every DAG has a source node

Lemma: If G is a DAG, then G has a node with no incoming edges (i.e., a source).

Proof. (by contradiction)

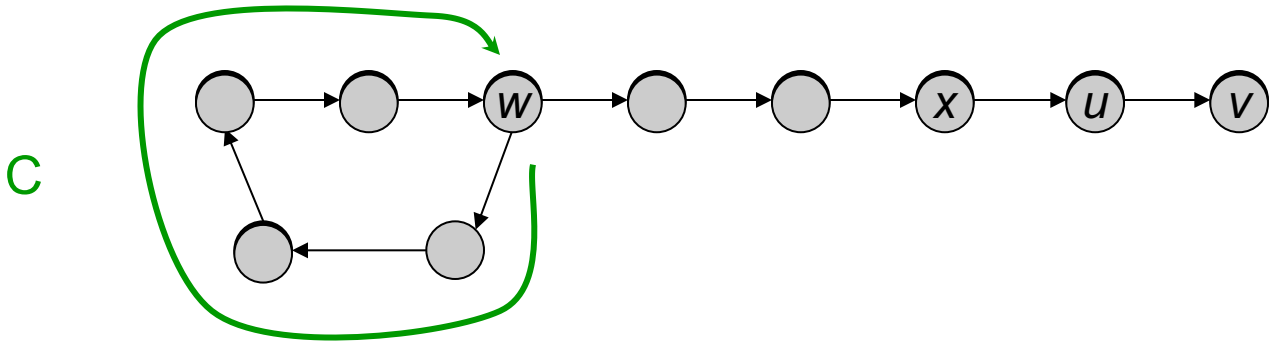
Suppose that G is a DAG and it has no source

Pick any node v, and begin following edges backward from v. Since v has at least one incoming edge (u, v) we can walk backward to u.

Then, since u has at least one incoming edge (x, u), we can walk backward to x.

Repeat until we visit a node, say w, twice.

Let C be the sequence of nodes encountered between successive visits to w. C is a cycle.



DAG => Topological Order

Lemma: If G is a DAG, then G has a topological order

Proof. (by induction on n)

Base case: true if $n = 1$.

Hypothesis: Every DAG with $n - 1$ vertices has a topological ordering.

Inductive Step: Given DAG with $n > 1$ nodes, find a source node v.

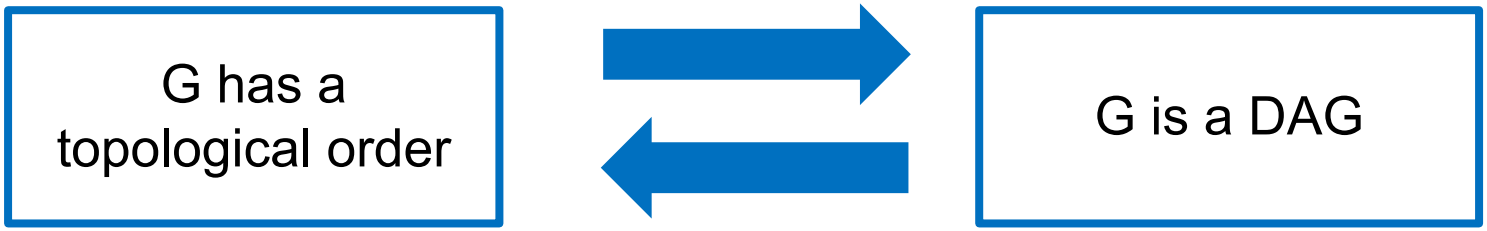
$G - \{ v \}$ is a DAG, since deleting v cannot create cycles.

Reminder: Always remove vertices/edges to use hypothesis

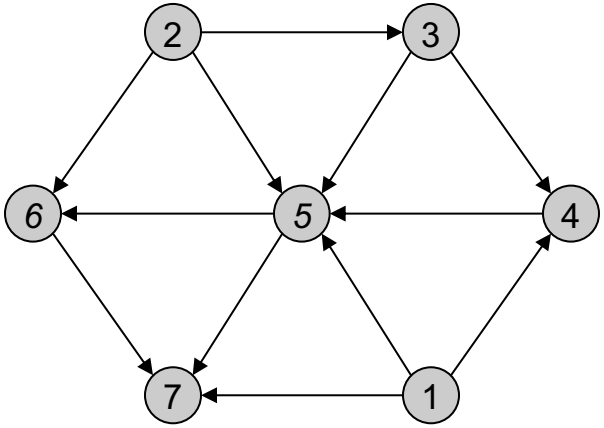
By hypothesis, $G - \{ v \}$ has a topological ordering.

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

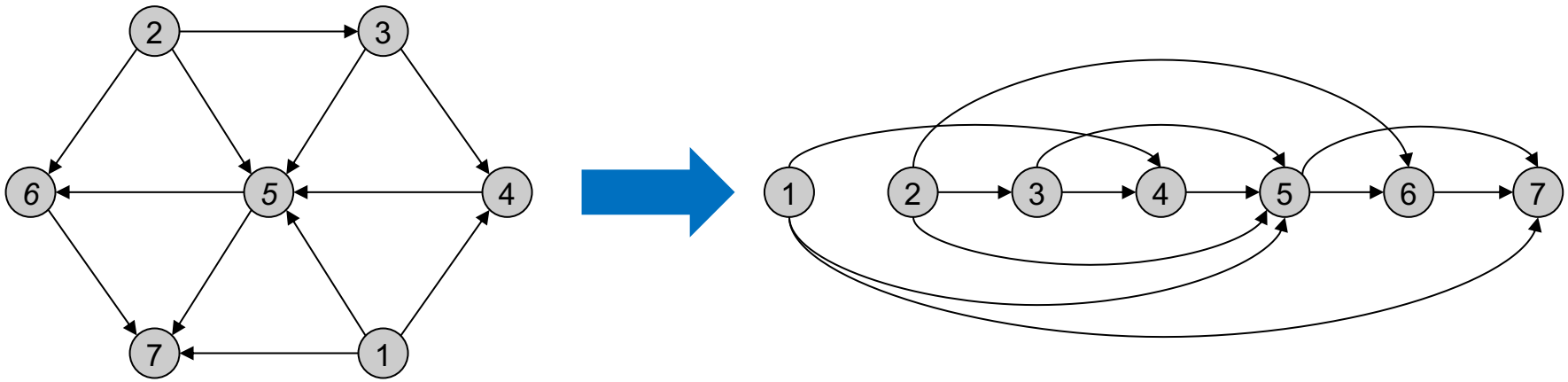
A Characterization of DAGs



Topological Order Algorithm 1: Example



Topological Order Algorithm 1: Example



Topological order: 1, 2, 3, 4, 5, 6, 7

Running time: O(n+m)

- Adjacency list
- Maintain # outgoing edge for each node