CSE221

Lecture 19:
Directed Graphs and Graph Algorithms

Acknowledgment: The content of this file is based on the slides of the textbook as well as the slides provided in former lectures at UNIST.

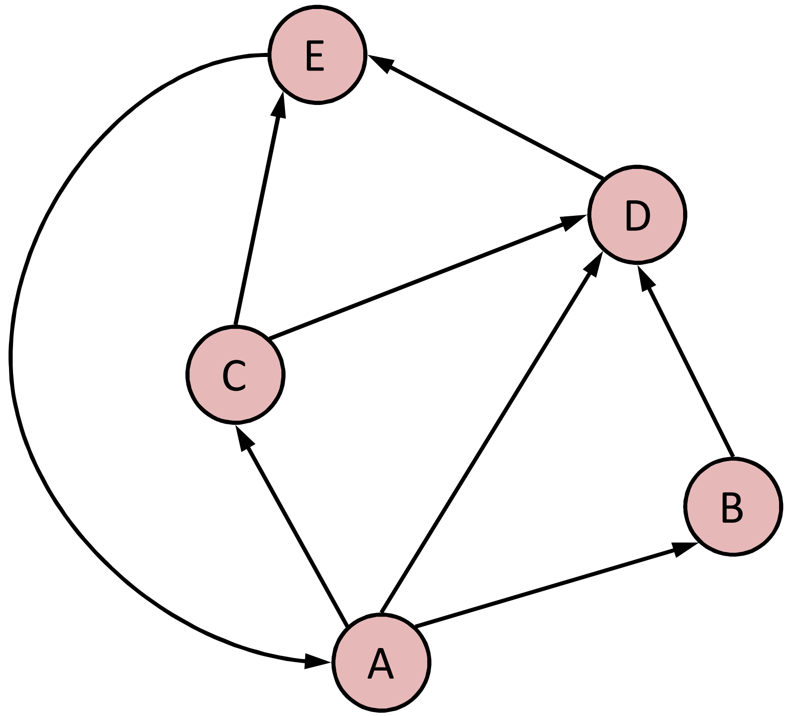
ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

Outline

- Directed graphs
  - Digraph properties
  - Reachability
  - Topological sorting
- Shortest path algorithms

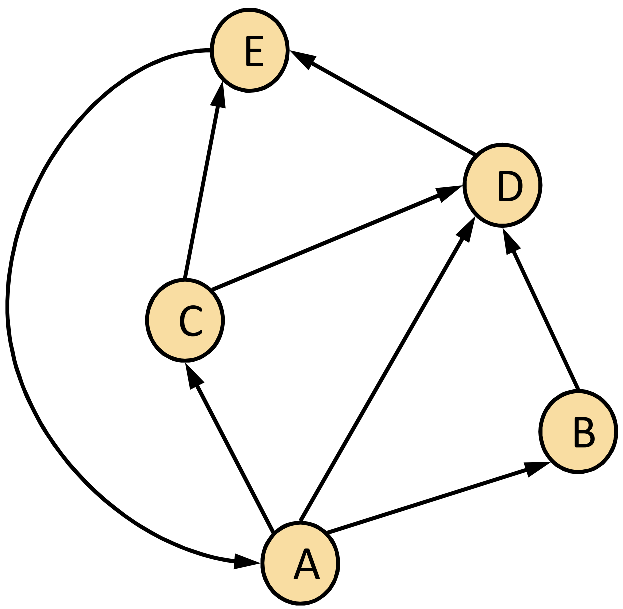
Digraphs

- A digraph is a graph whose edges are all directed
  - Short for "directed graph"
- Applications
  - flights
  - one-way streets
  - work decompositions



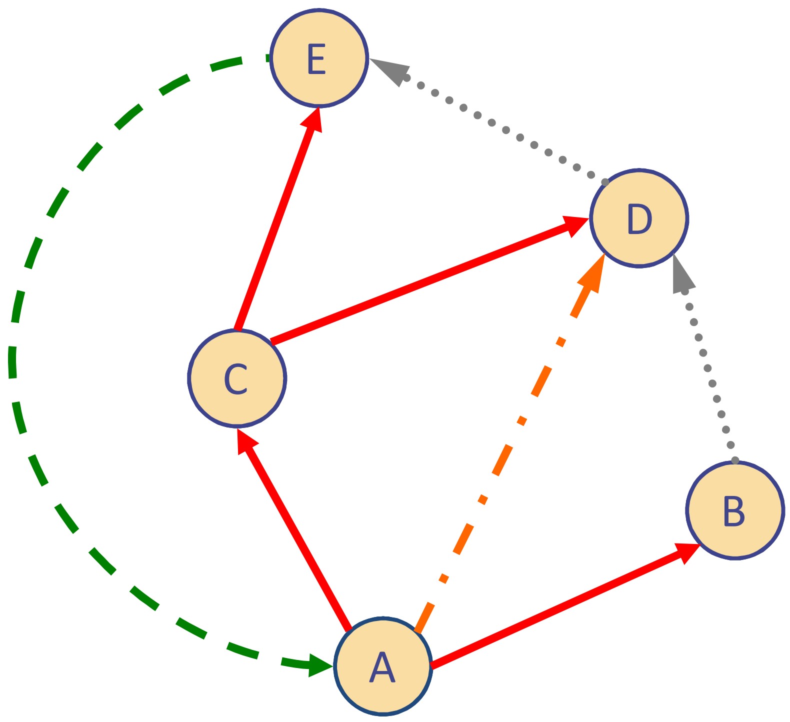
Digraph Properties

- A graph G=(V,E) such that
  - Each edge goes in one direction
  - Edge (a,b) goes from a to b, but not b to a
- If G is simple, $m \leq n\,(n - 1)$
- If we keep in-edges and out-edges in separate adjacency lists
  - We can perform listing of incoming edges and outgoing edges in time proportional to their size



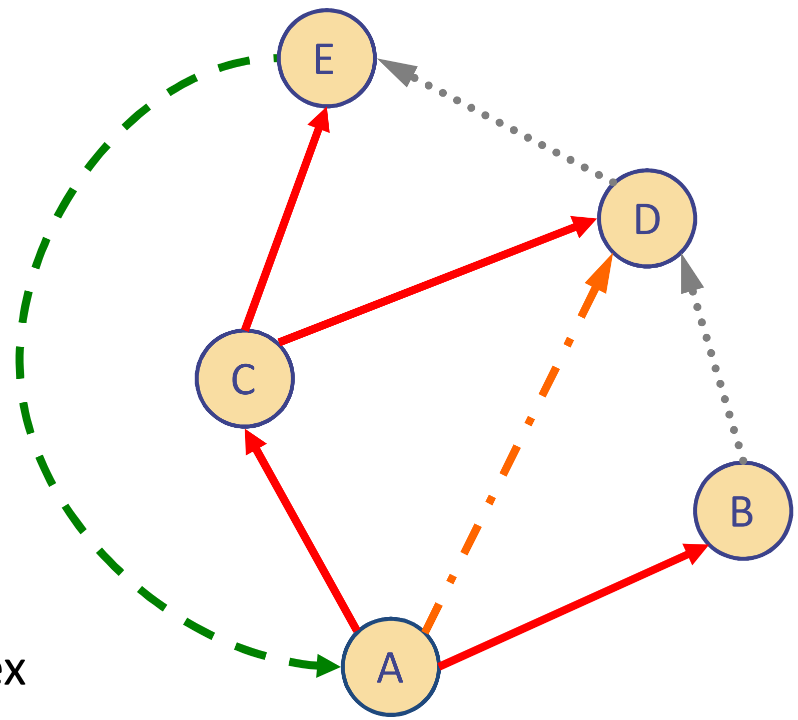
Directed DFS

- DFS traversing edges only along directions in digraphs
- A directed DFS starting at a vertex s determines the vertices reachable from s



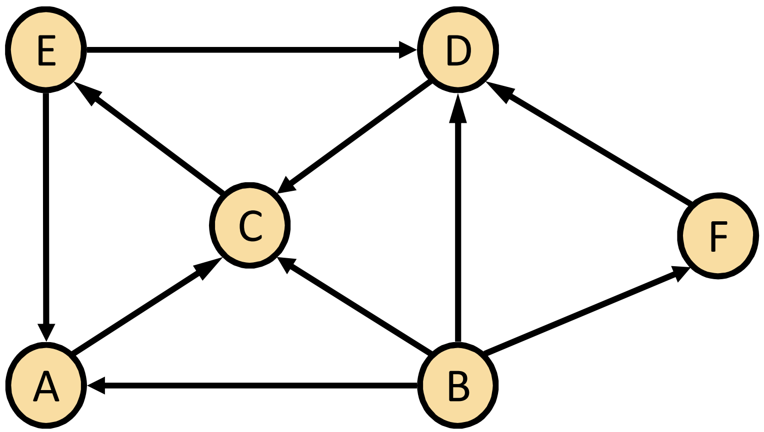
Directed DFS

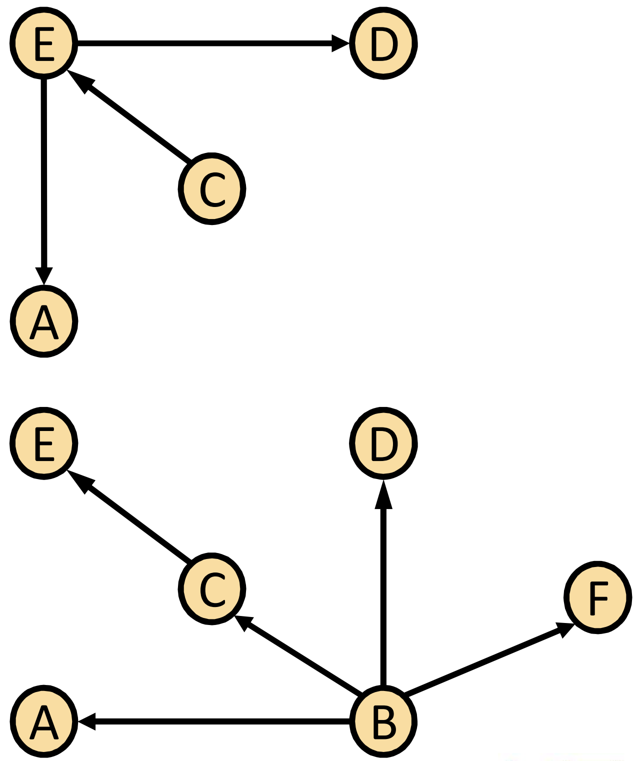
- In the directed DFS algorithm, we have four types of edges
  - discovery edges
    - Tree edges
  - back edges
    - which connect a vertex to an ancestor in the DFS tree
  - forward edges
    - which connect a vertex to a descendent in the DFS tree
  - cross edges
    - which connect a vertex to a vertex that is neither its ancestor nor its descendent



Reachability

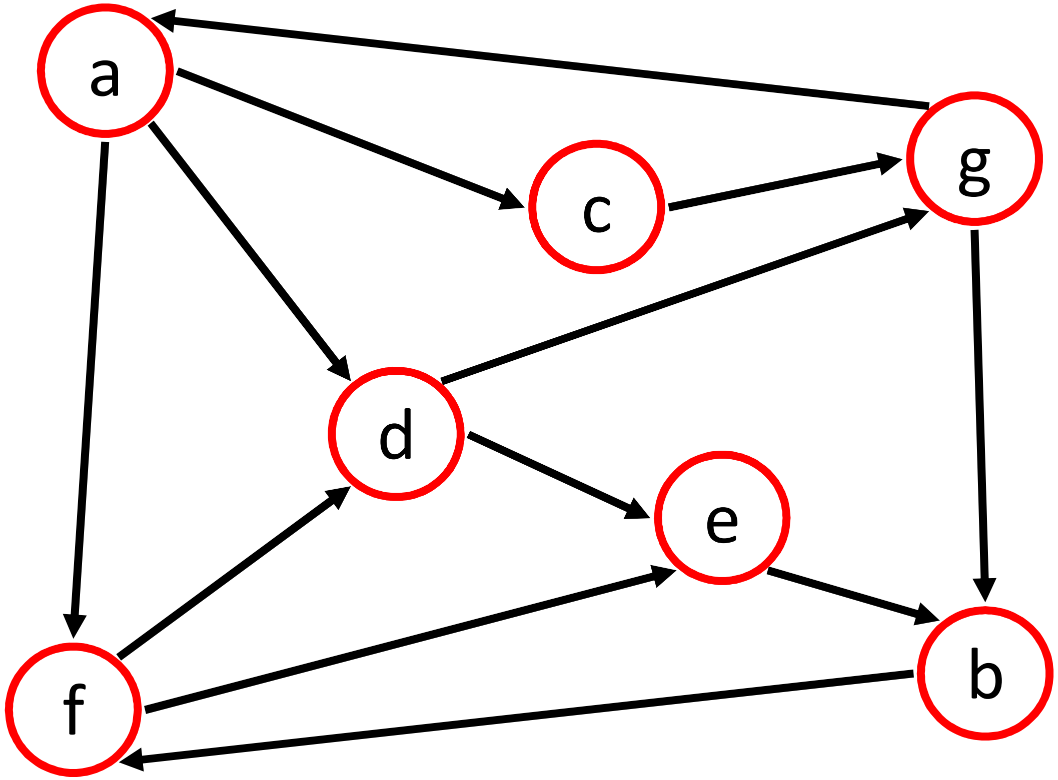
- DFS tree rooted at v: vertices reachable from v via directed paths





Strong Connectivity

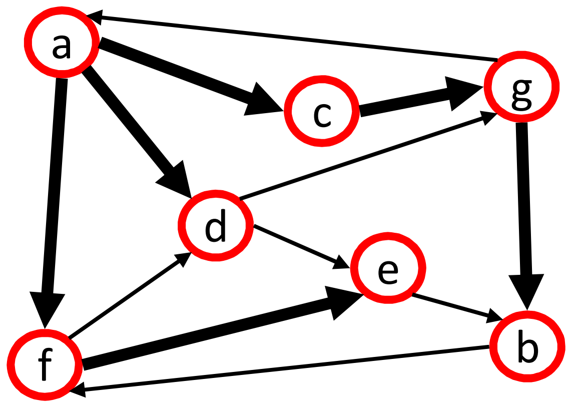
- Each vertex can reach all other vertices
  - Run directed DFS for every vertex : O(n(n+m))



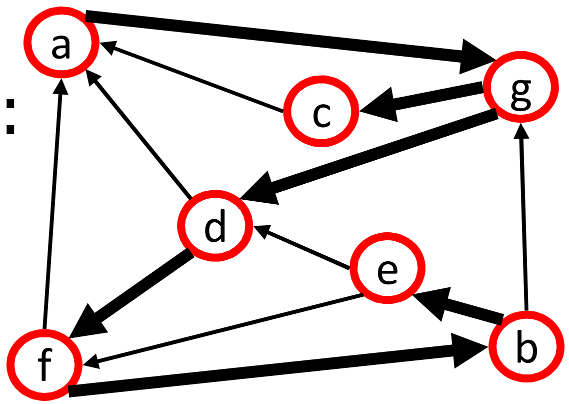
Strong Connectivity Algorithm

- Pick any vertex v in G
- Perform a DFS from v in G
  - If there's a w not visited, print "no"
- Let G′ be G with edges reversed
- Perform a DFS from v in G′
  - If there's a w not visited, print "no"
  - Else, print "yes"
    - Each of the vertices visited can reach v
- Running time: O(n+m)
  - Requires only two directed DFS
    - If every node can be reached from a vertex v, and every node can reach v, then the graph is strongly connected.

G:



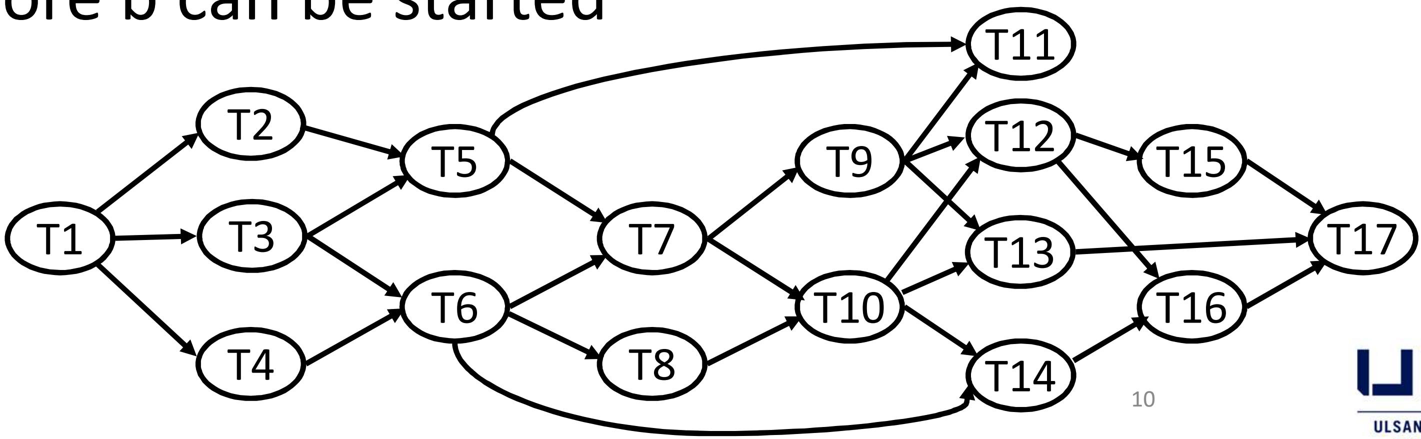
G' :



ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

DAGs and Topological Ordering

- Decompose work into tasks that can be executed
- Conceptualize tasks and ordering as task dependency DAG (directed acyclic graph)
  - vertex = task
  - edge = control dependency
- Scheduling: edge (a,b) means task a must be completed before b can be started



DAGs and Topological Ordering

- A directed acyclic graph (DAG) is a digraph that has no directed cycles
- A topological ordering of a digraph is a numbering

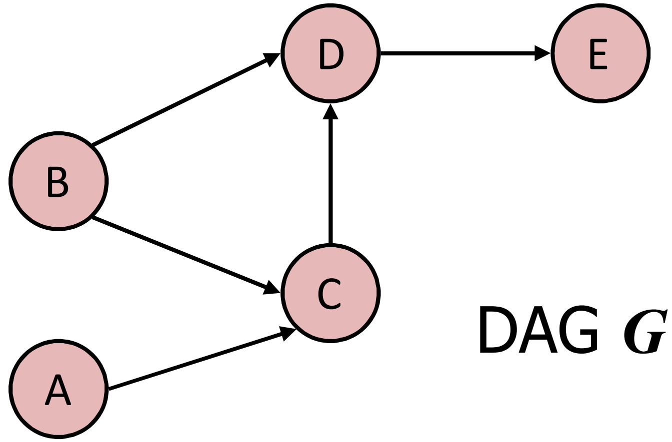
$$v_1, \dots, v_n$$

of the vertices such that for every edge (v_i, v_j), we have $i < j$

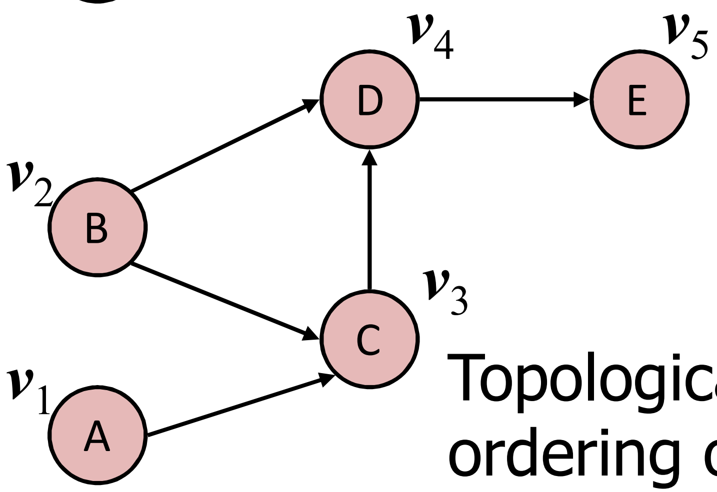
- In a task scheduling digraph, a topological ordering is a task sequence that satisfies the control dependency across all tasks

Theorem

A digraph admits a topological ordering if and only if it is a DAG



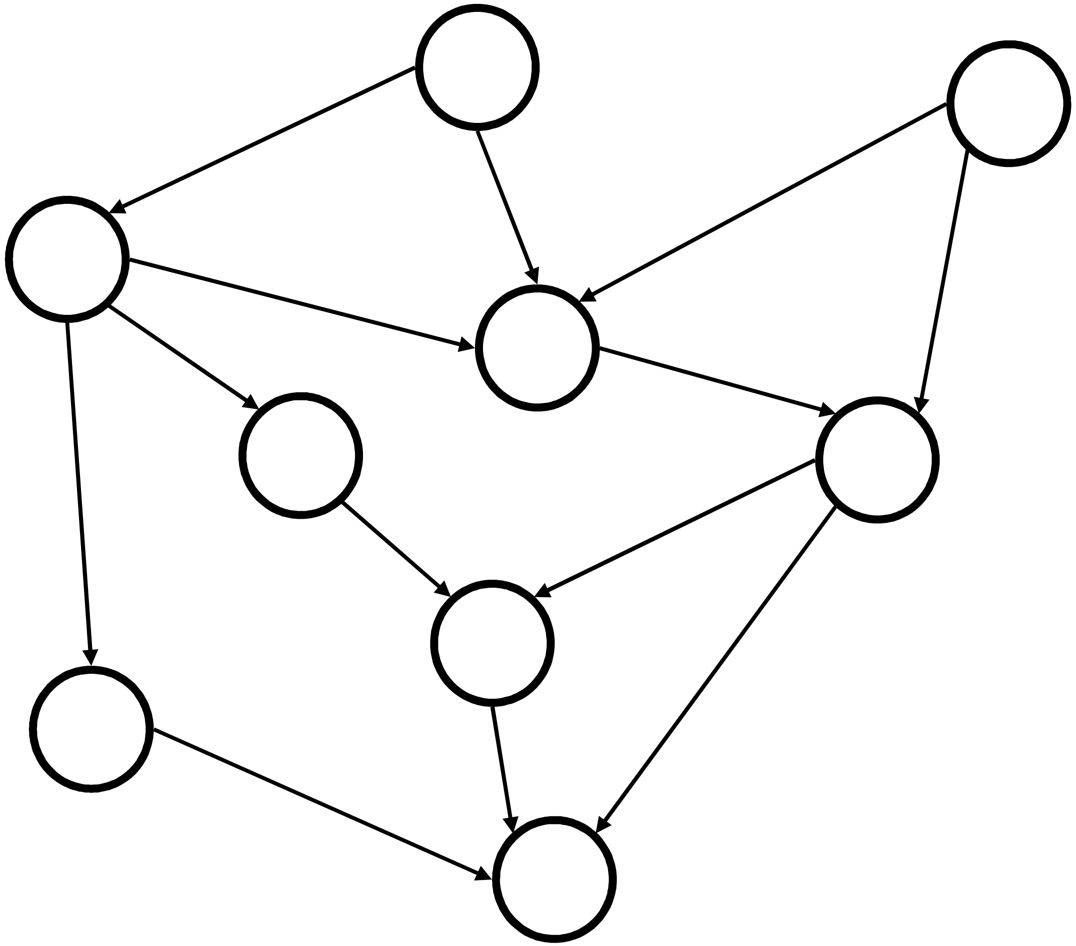
DAG G



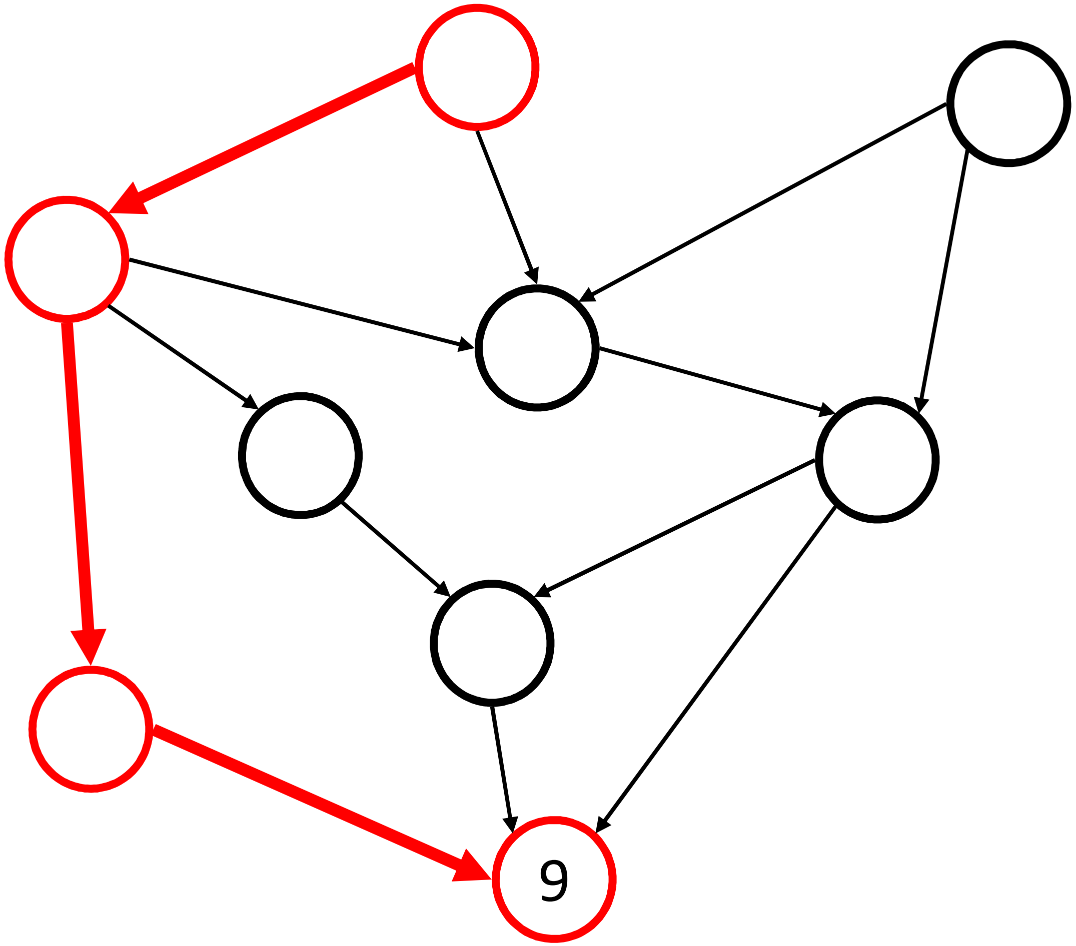
Topological ordering of G

Q: is topological sorting unique?

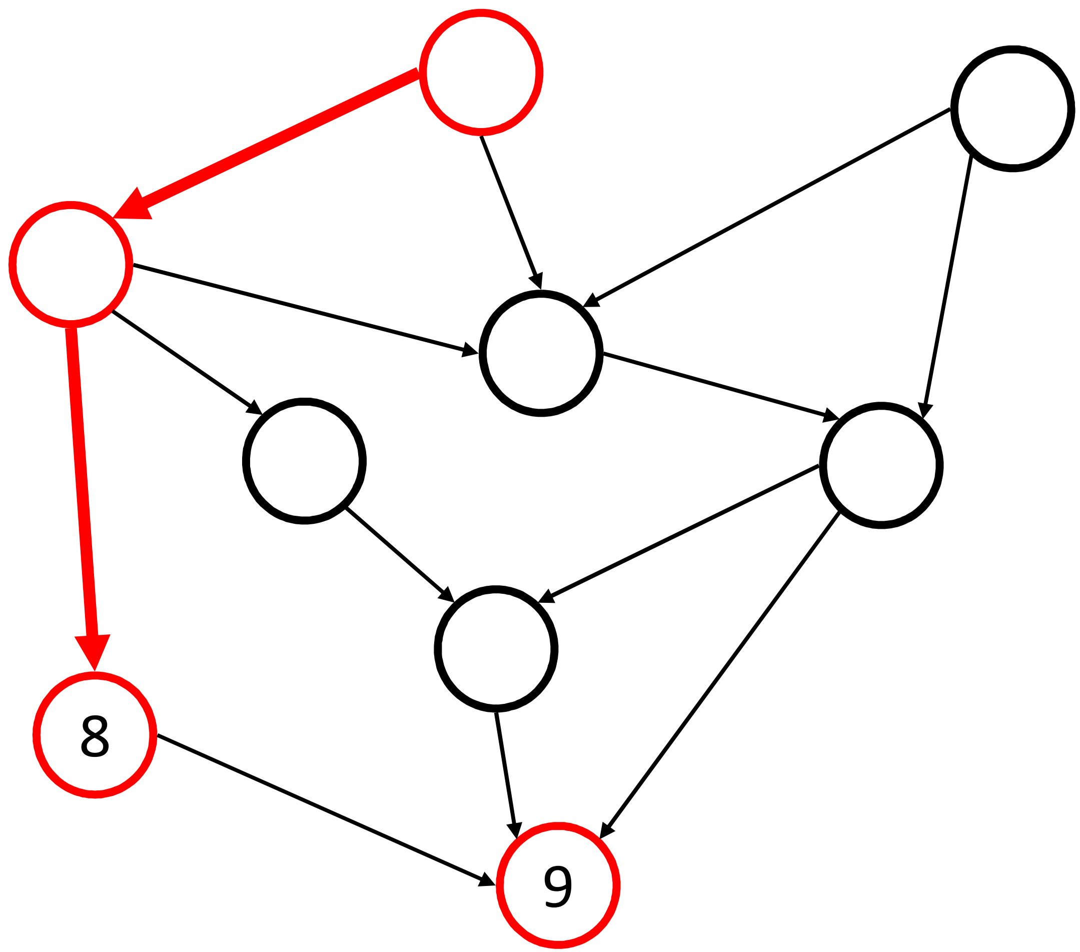
Topological Sorting Example



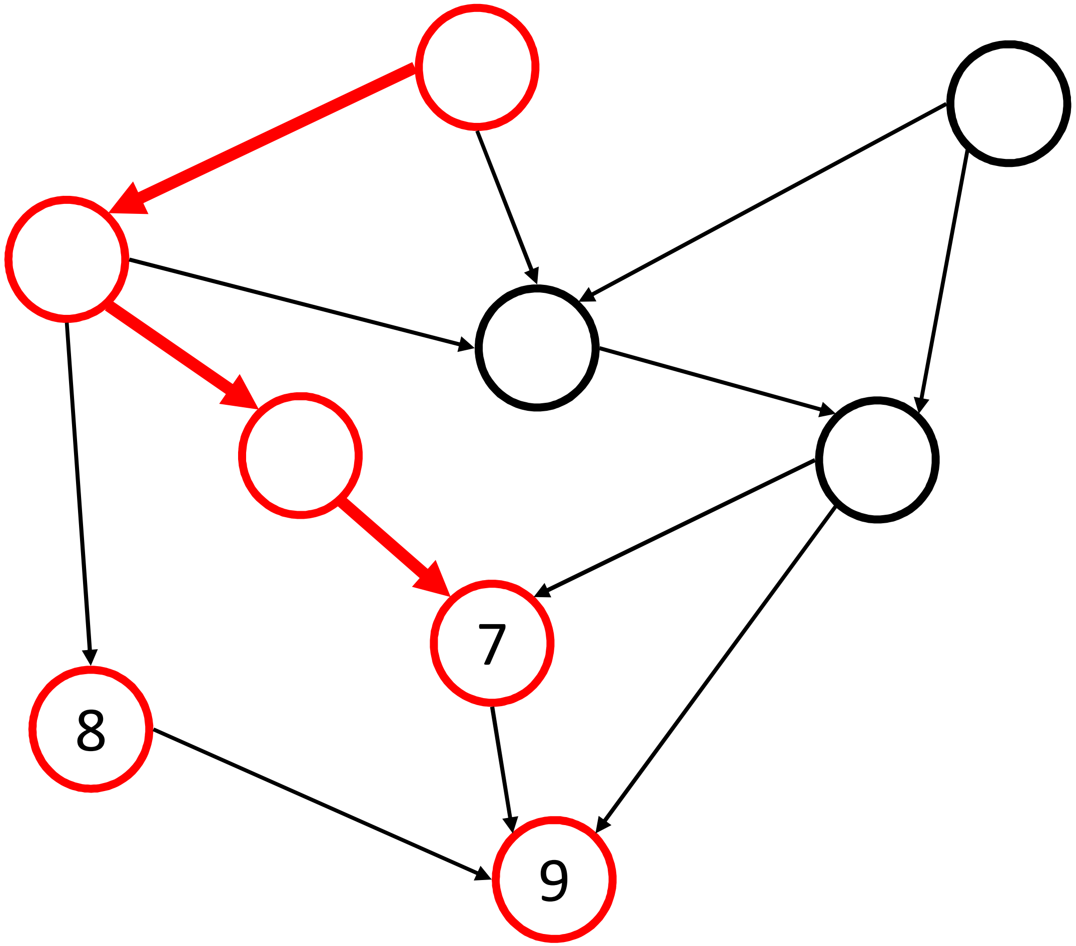
Topological Sorting Example



Topological Sorting Example

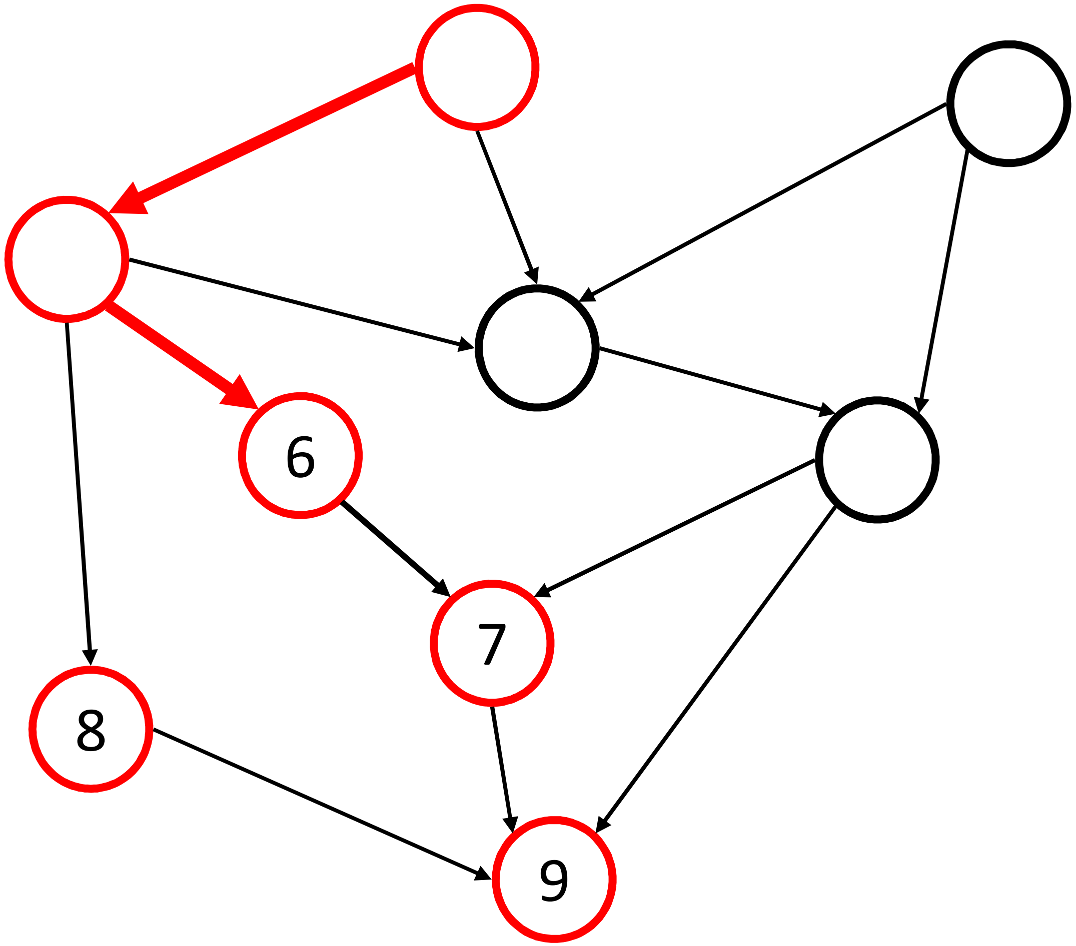


Topological Sorting Example

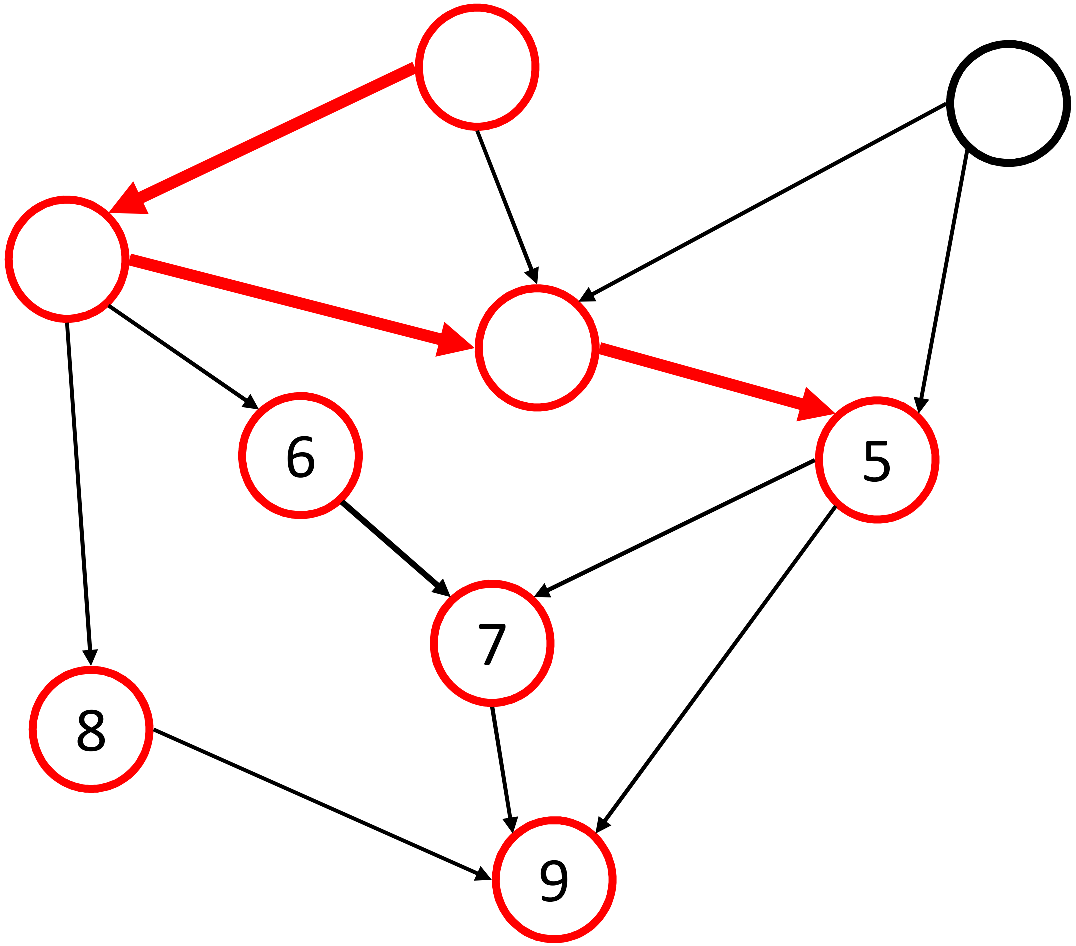


ULSAN NATIONAL INSTITUTE OF SCIENCE AND TECHNOLOGY

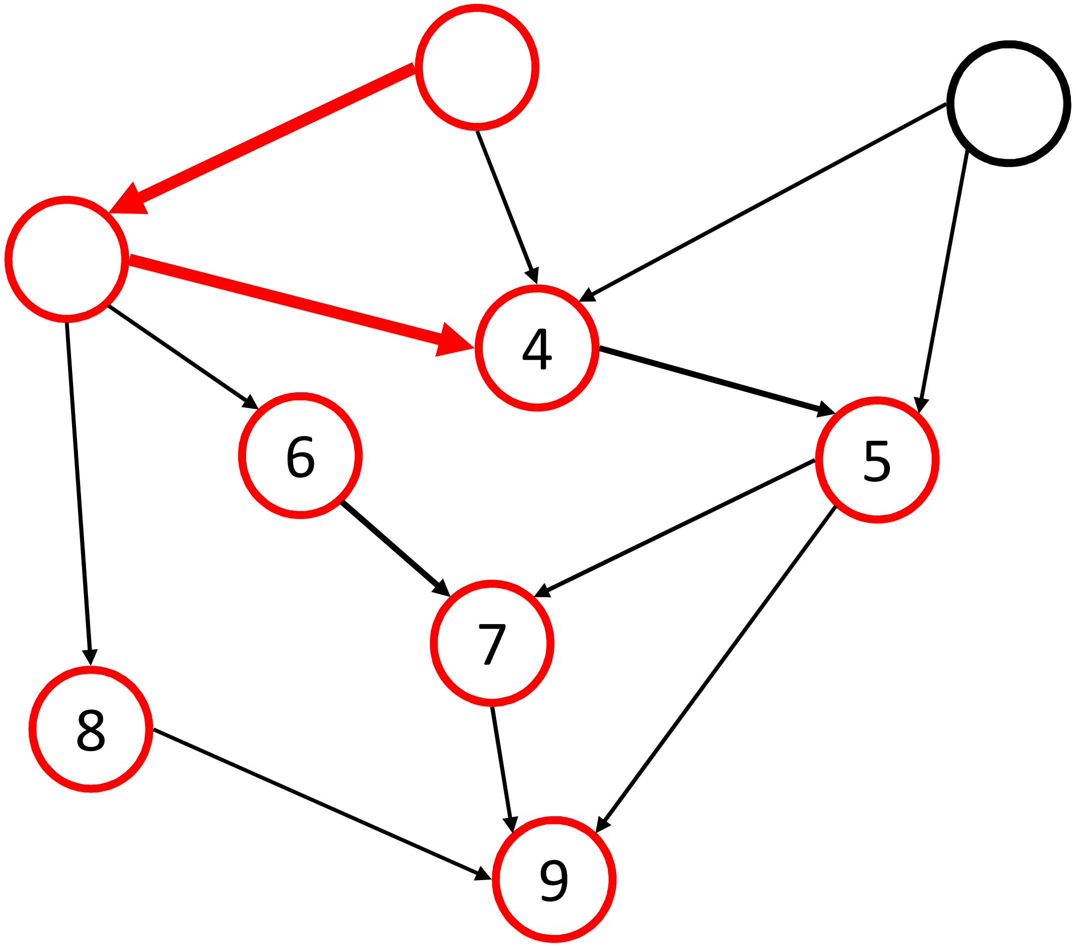
Topological Sorting Example



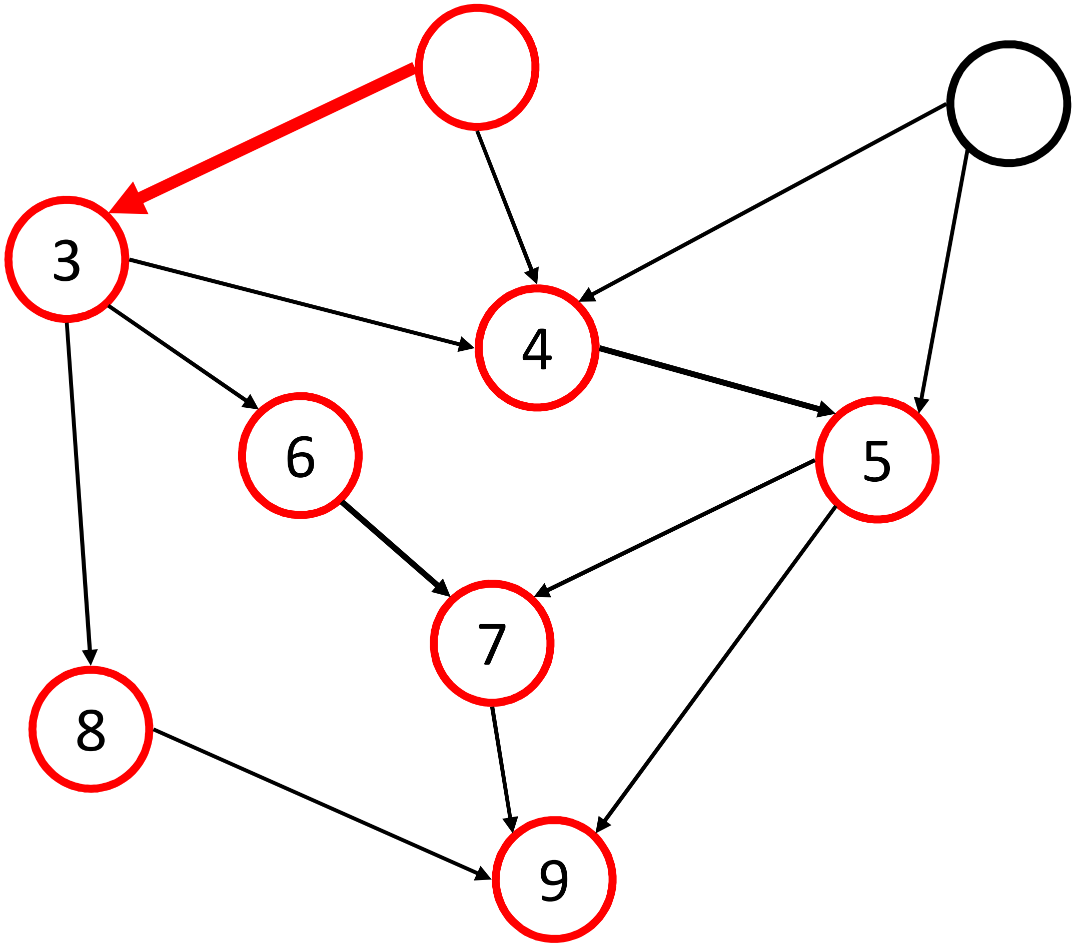
Topological Sorting Example



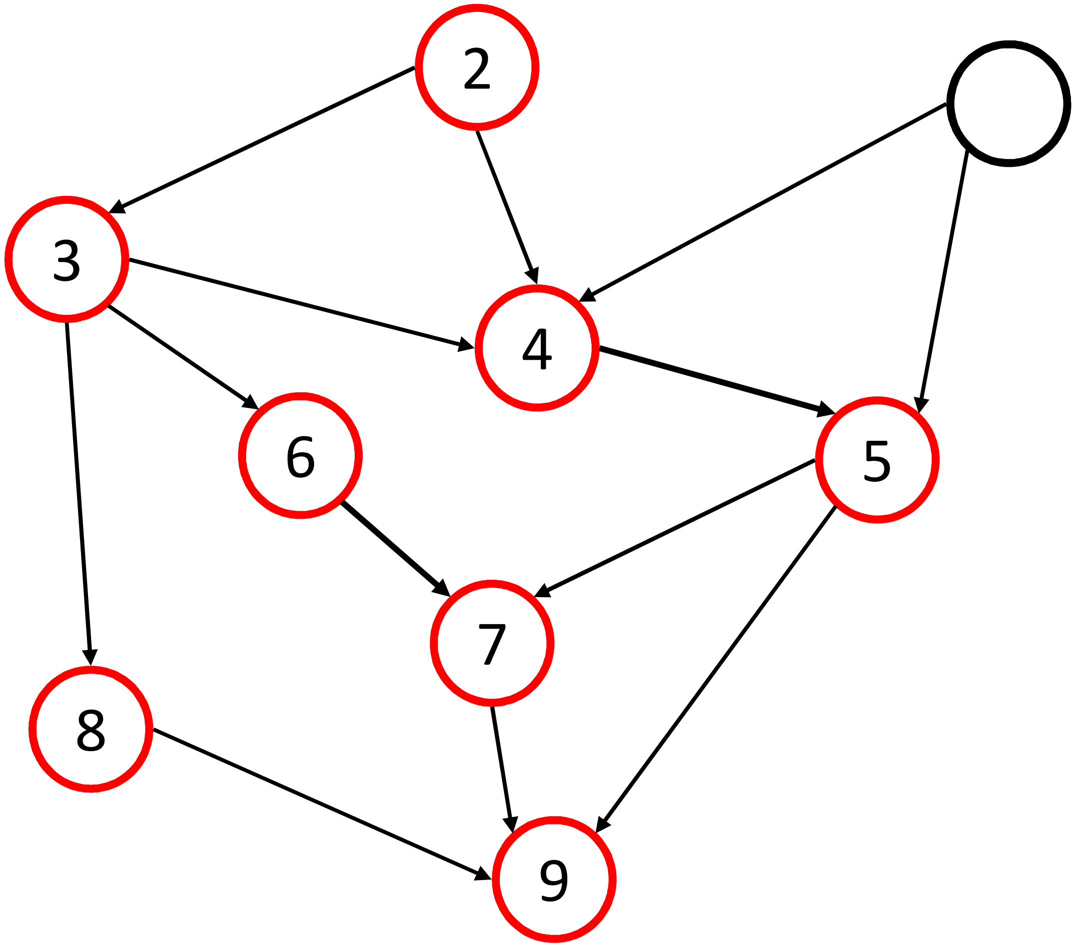
Topological Sorting Example



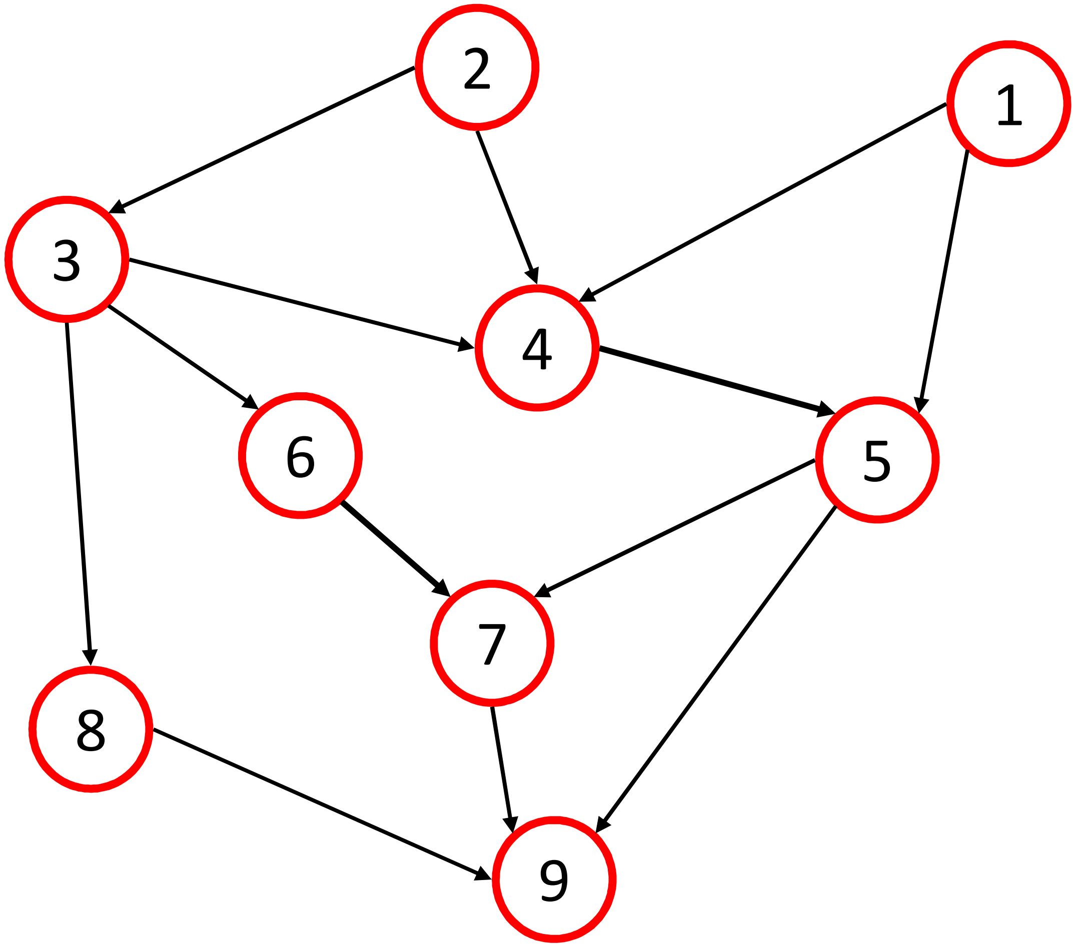
Topological Sorting Example



Topological Sorting Example



Topological Sorting Example



Algorithm for Topological Sorting

Algorithm TopologicalSort(G)
    $H \leftarrow G$ // Temporary copy of G
    $n \leftarrow G.numVertices()$
    while H is not empty **do**
        Let v be a vertex with no outgoing edges
        Label $v \leftarrow n$
        $n \leftarrow n - 1$
        Remove v from H

Implementation with DFS

- Simulate the algorithm by using depth-first search
- O(n+m) time

```
Algorithm topologicalDFS(G)
   Input dag G
   Output topological ordering of G

   n ← G.numVertices()
   for all u in G.vertices()
      u.setLabel(UNEXPLORED)
   for all v in G.vertices()
      if v.getLabel() = UNEXPLORED
         topologicalDFS(G, v)
```

```
Algorithm topologicalDFS(G, v)
   Input graph G and a start vertex v of G
   Output labeling of the vertices of G
      in the connected component of v

   v.setLabel(VISITED)
   for all e in v.outEdges()
      { outgoing edges }
      w ← e.opposite(v)
      if w.getLabel() = UNEXPLORED
         { e is a discovery edge }
         topologicalDFS(G, w)
      else
         { e is a forward or cross edge }
   Label v with topological number n
   n ← n - 1
```