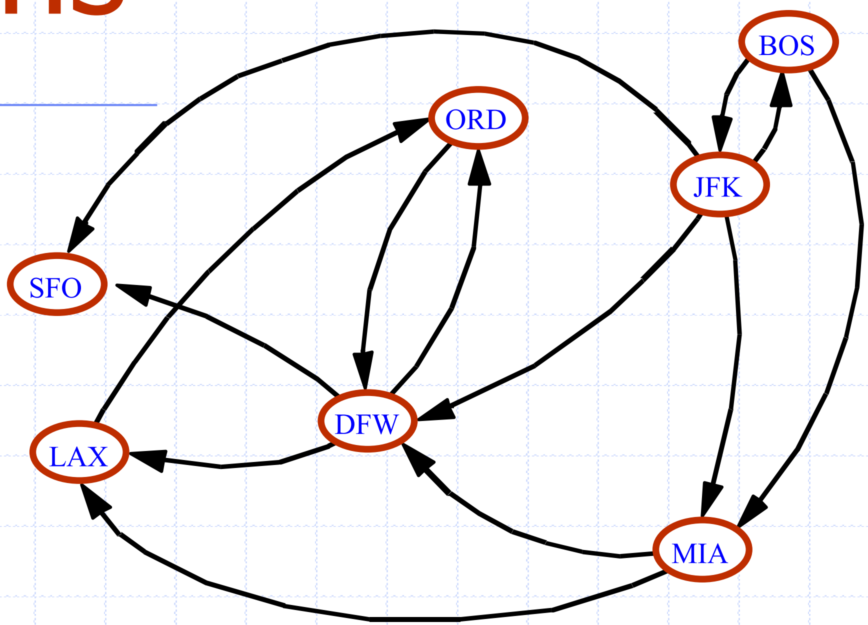
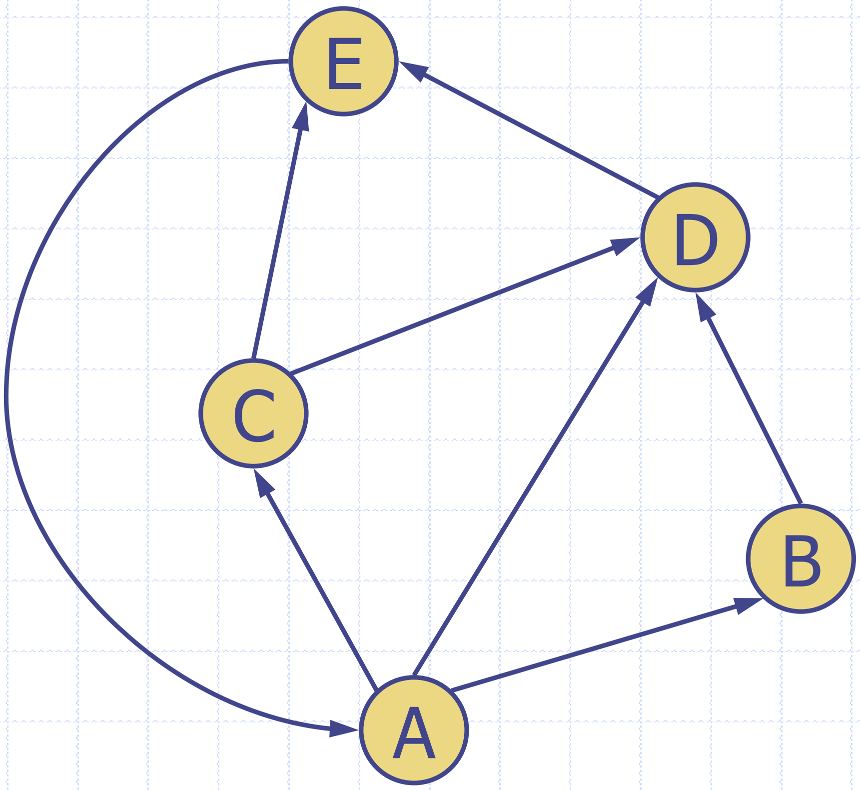
Presentation for use with the textbook Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014 With notes from R. Solis-Oba

Directed Graphs

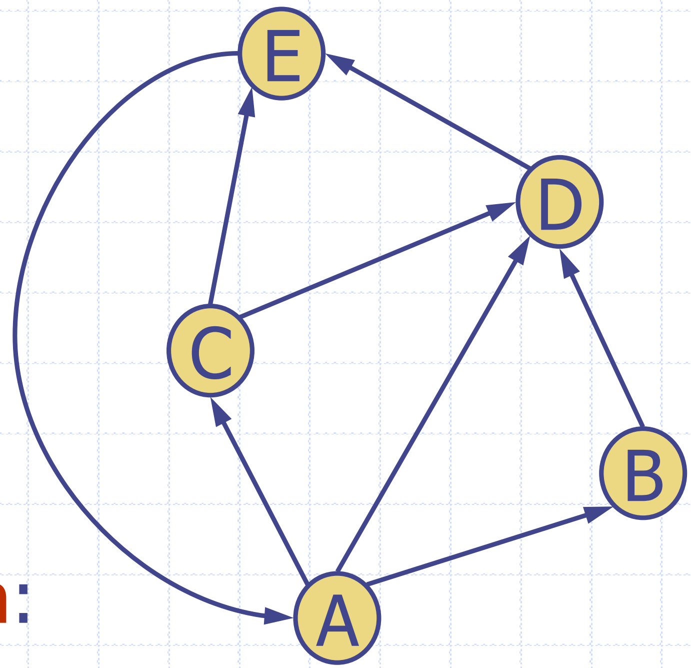


Digraphs

- A digraph is a graph whose edges are all directed
  - Short for “directed graph”
- Applications
  - one-way streets
  - flights
  - task scheduling



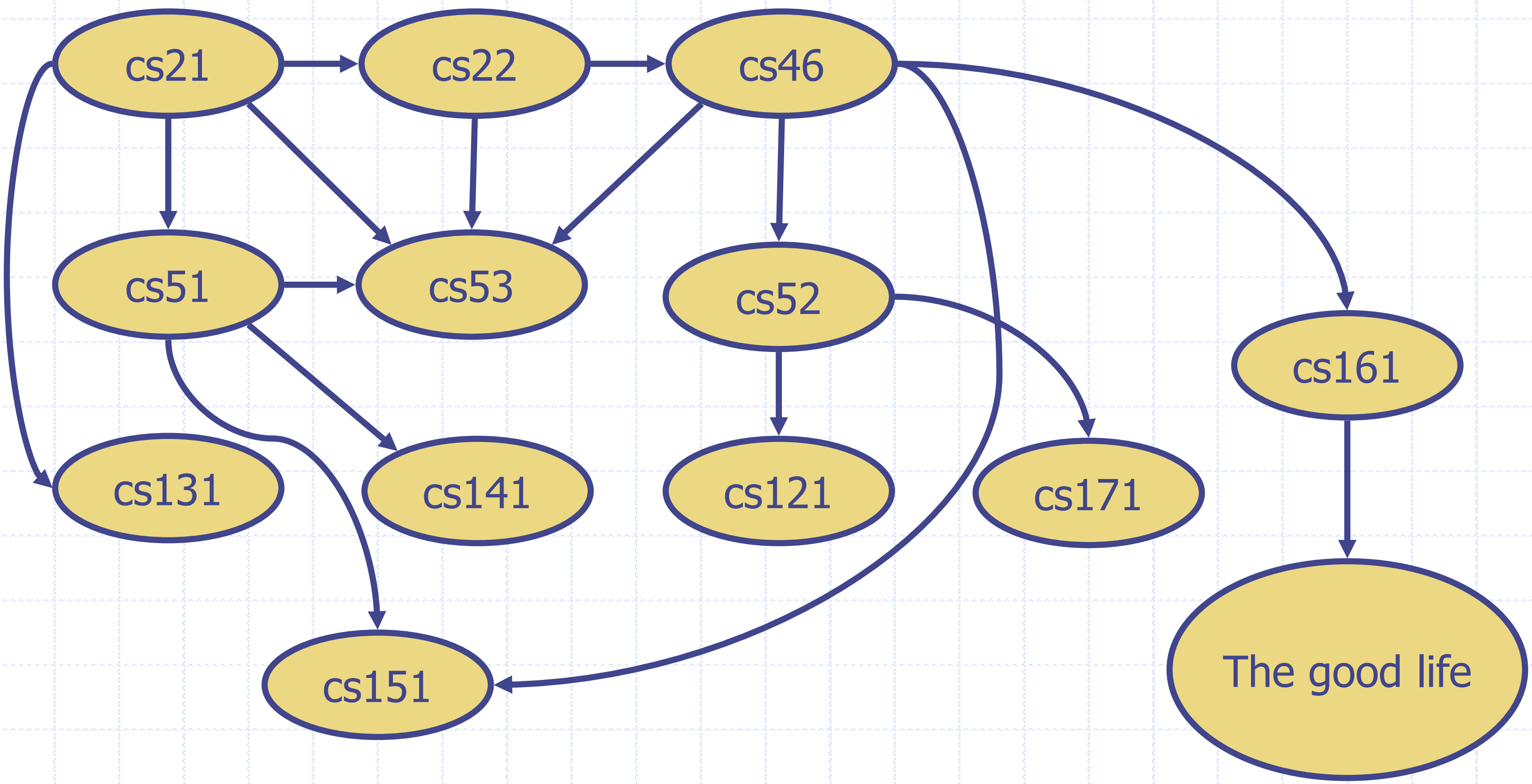
Digraph Properties



- 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, $m \le n \cdot (n - 1)$
- 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

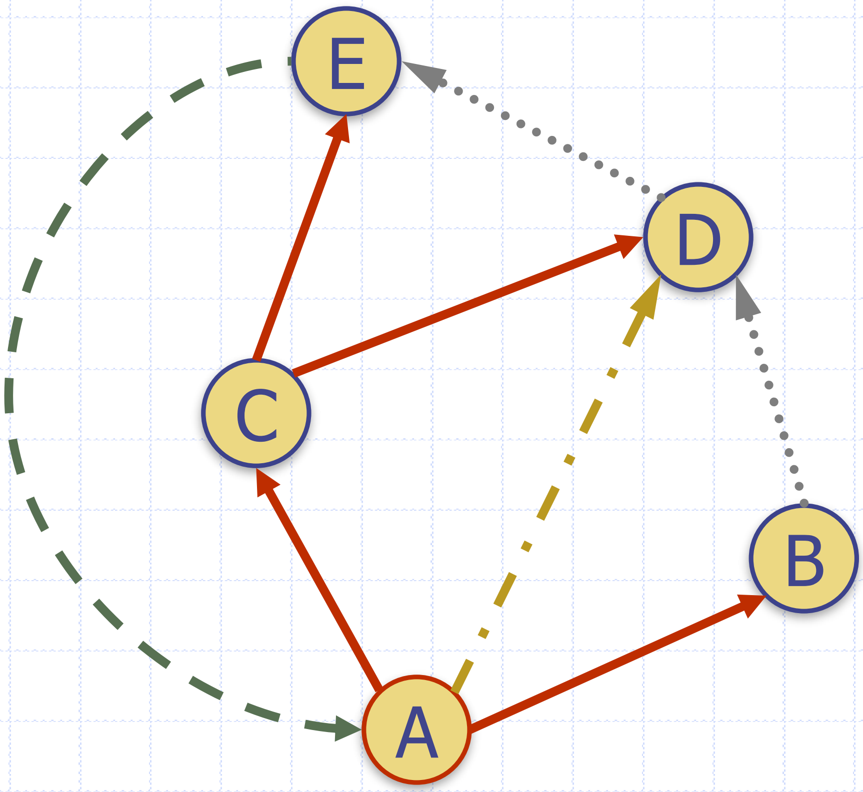
Digraph Application

- Scheduling: edge (a,b) means task a must be completed before b can be started



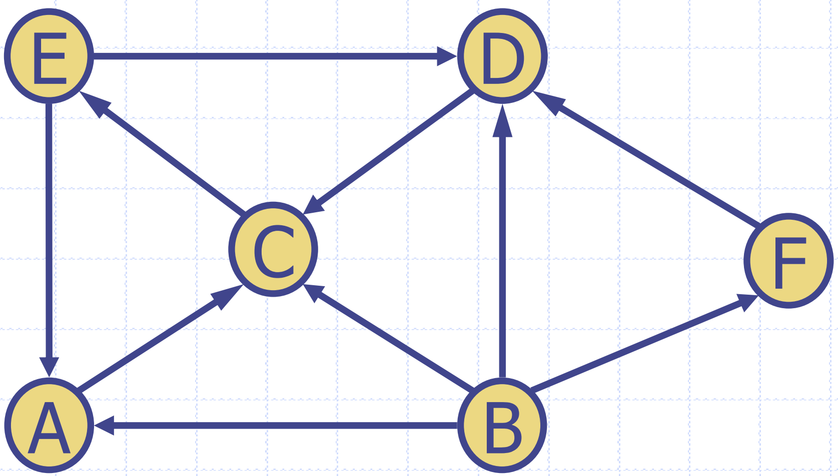
Directed DFS

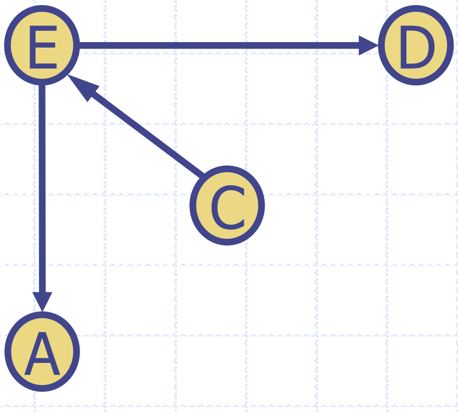
- We can specialize the traversal algorithms (DFS and BFS) to digraphs by traversing edges only along their direction
- A directed DFS starting at a vertex s determines the vertices reachable from s

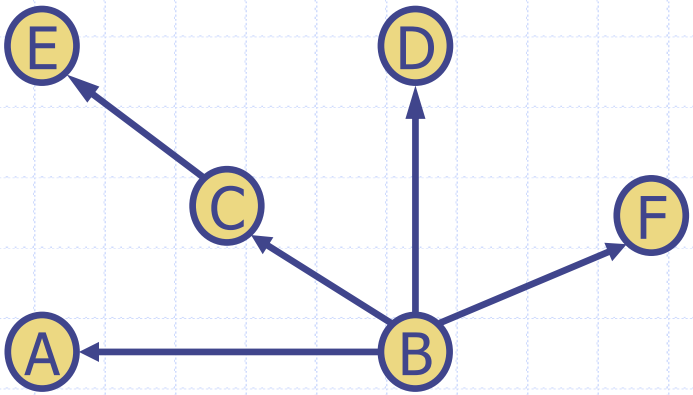


Reachability

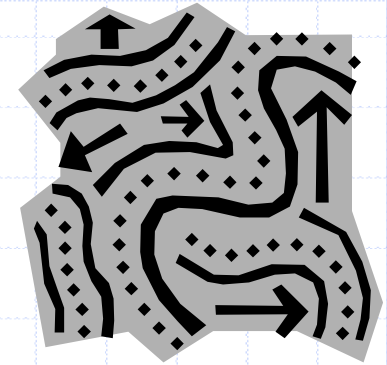
- DFS tree rooted at v: vertices reachable from v via directed paths



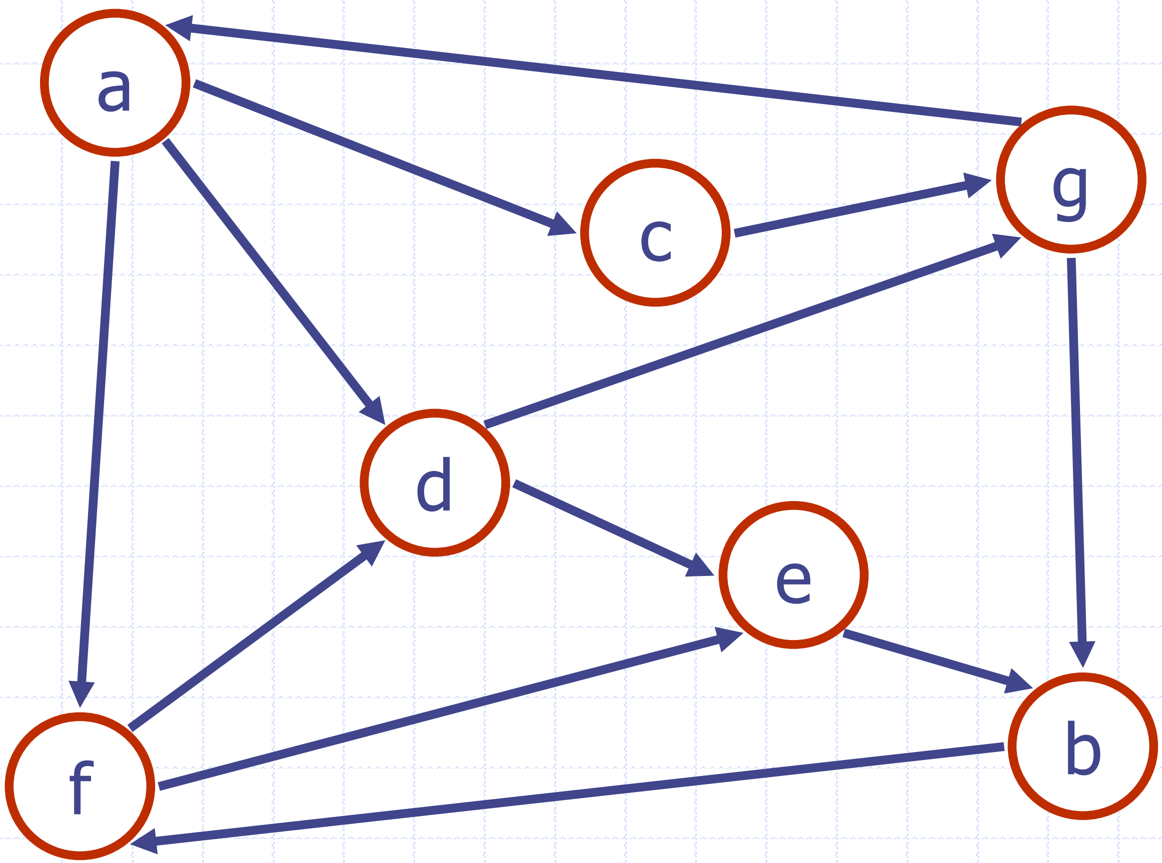




Strong Connectivity



- Each vertex can reach all other vertices

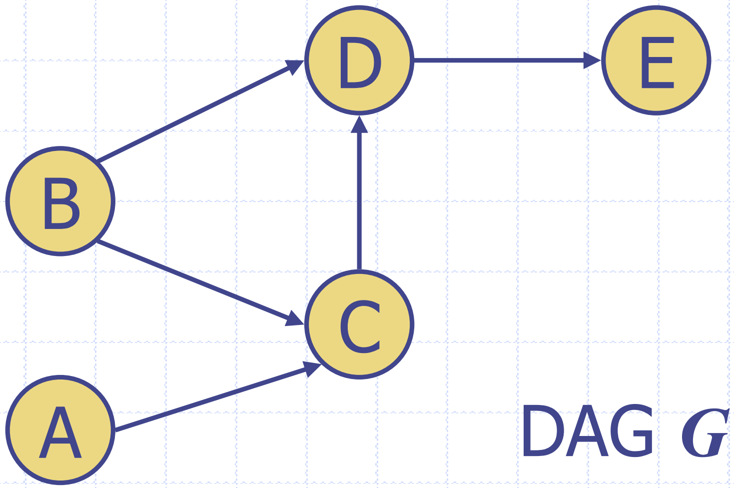


DAGs and Topological Ordering

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

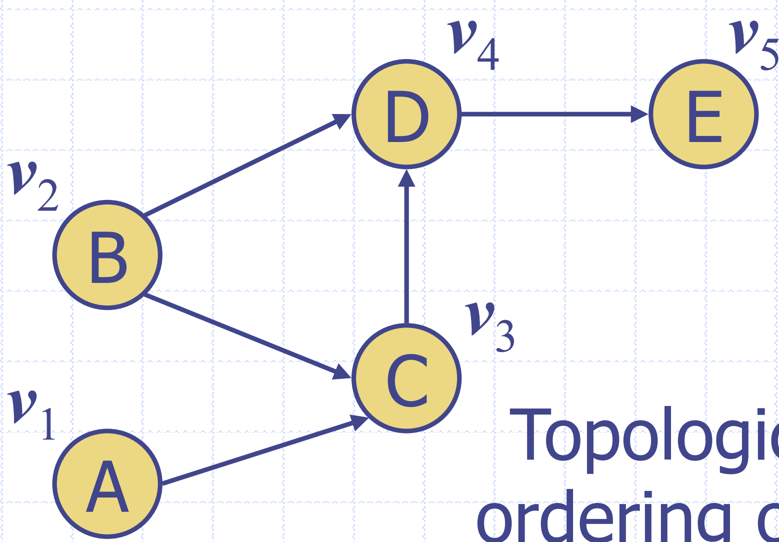
  $$v_1 , \ldots, v_n$$

  of the vertices such that for every edge (v_i , v_j), we have $i < j$
- Example: in a task scheduling digraph, a topological ordering a task sequence that satisfies the precedence constraints



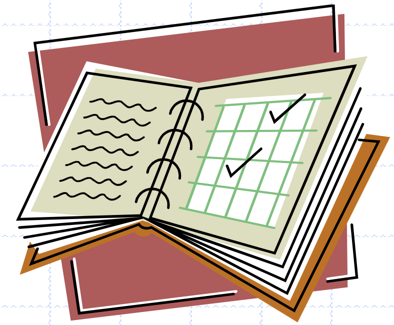
DAG G

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

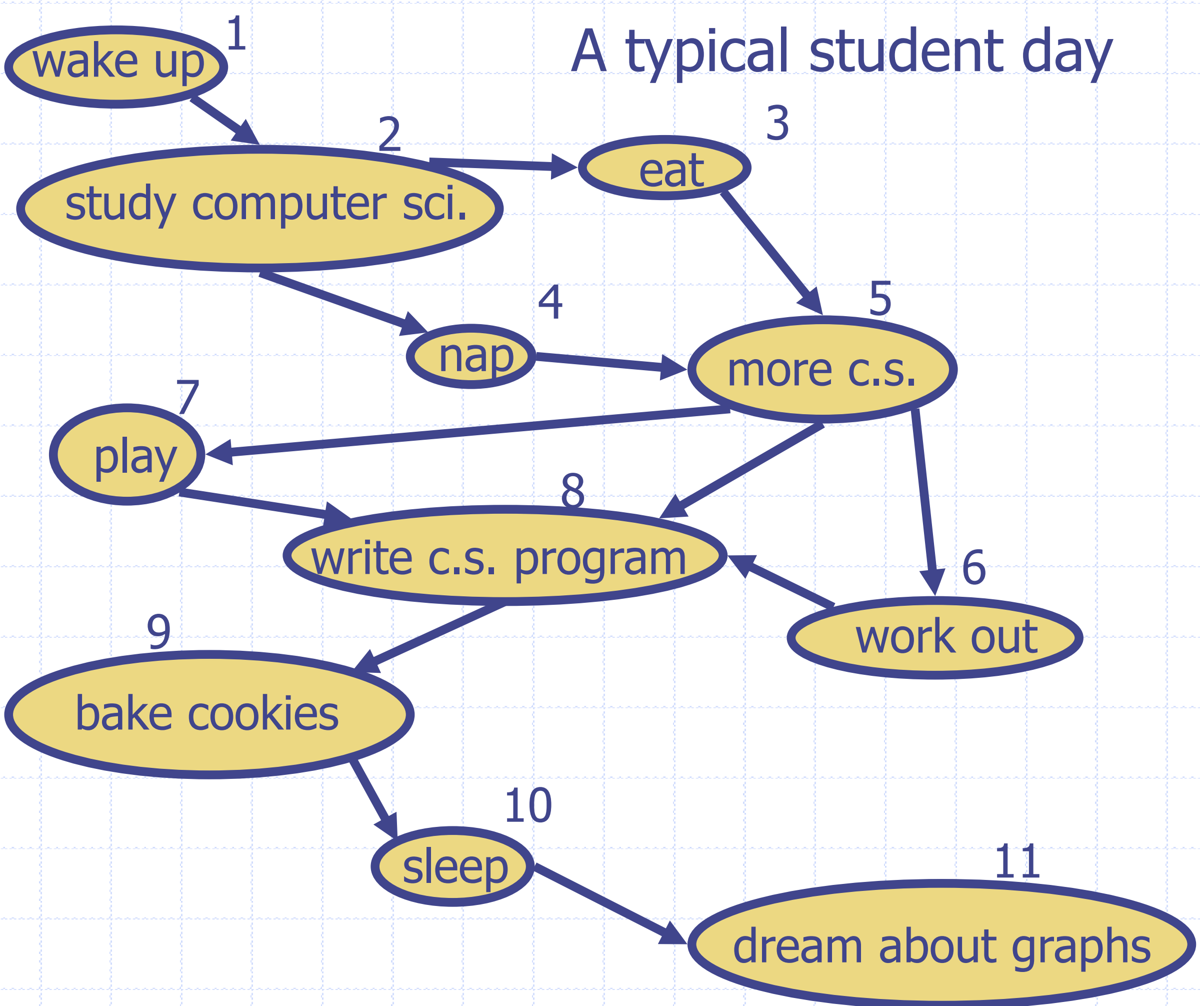


Topological ordering of G

Topological Sorting



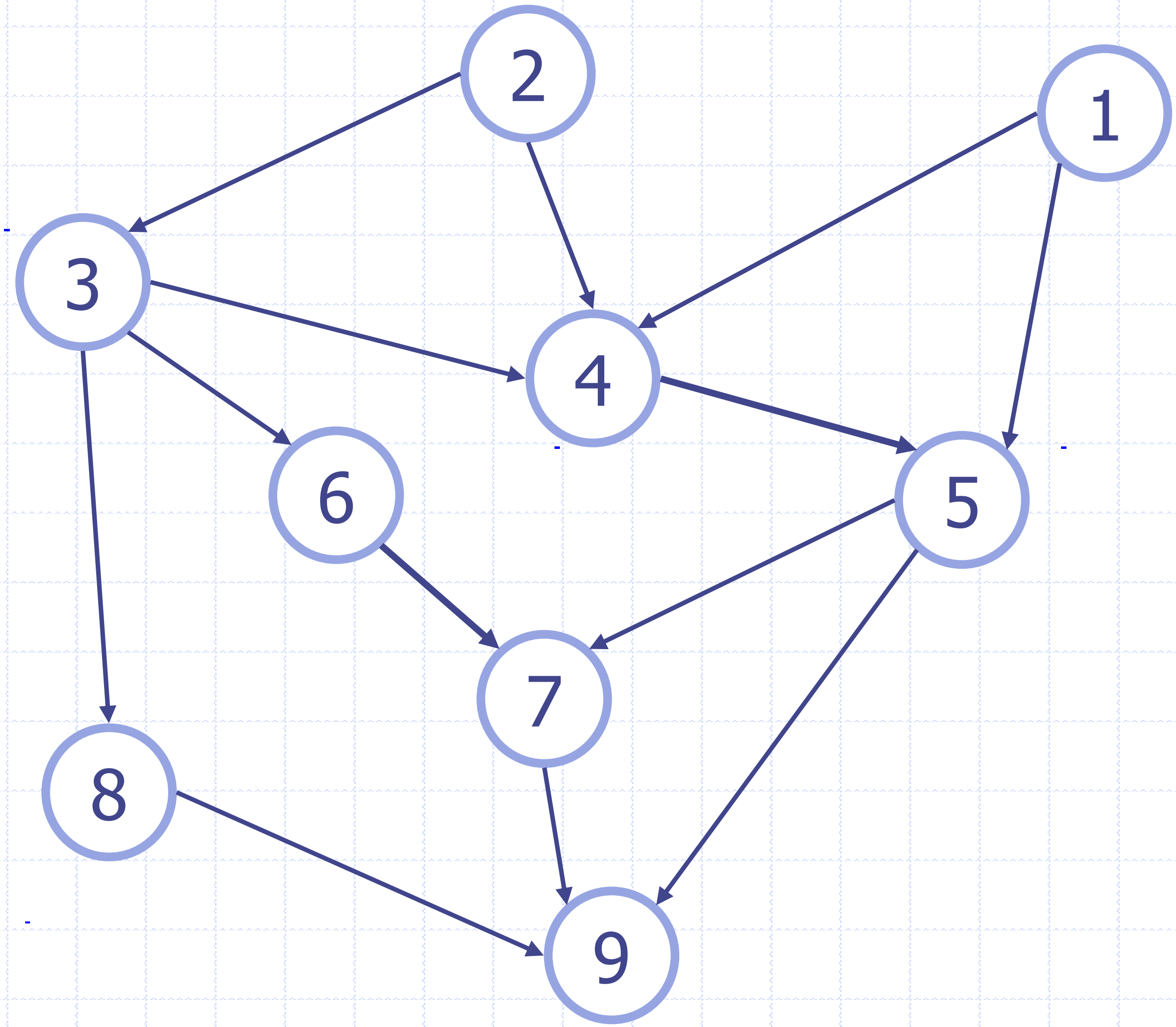
- Number vertices, so that (u,v) in E implies u < v



Algorithm for Topological Sorting

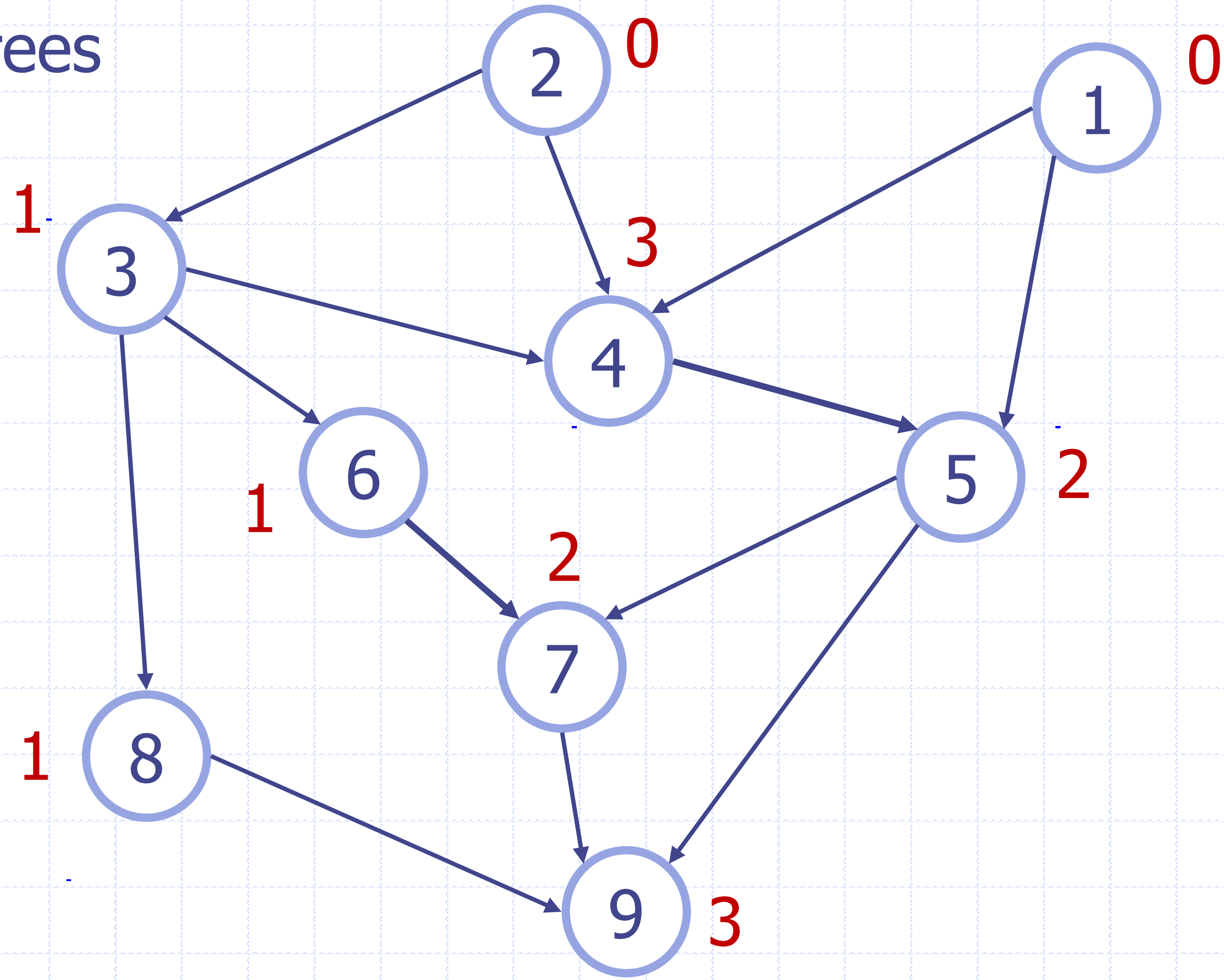
```
Algorithm TopologicalOrdering(G)
In: Directed graph G
Out: Topological ordering for the vertices of G
Q ← empty queue
for each vertex u of G do {
    u.inDegree ← in-degree of u
    if u.inDegree = 0 then Q.enqueue(u)
}
while Q is not empty do {
    u ← Q.dequeue()
    print (u)
    for each outgoing edge (u,v) incident on u do {
        v.inDegree ← v.inDegree – 1
        if v.inDegree = 0 then Q.enqueue (v)
    }
}
```

Topological Sorting Example



Topological Sorting Example

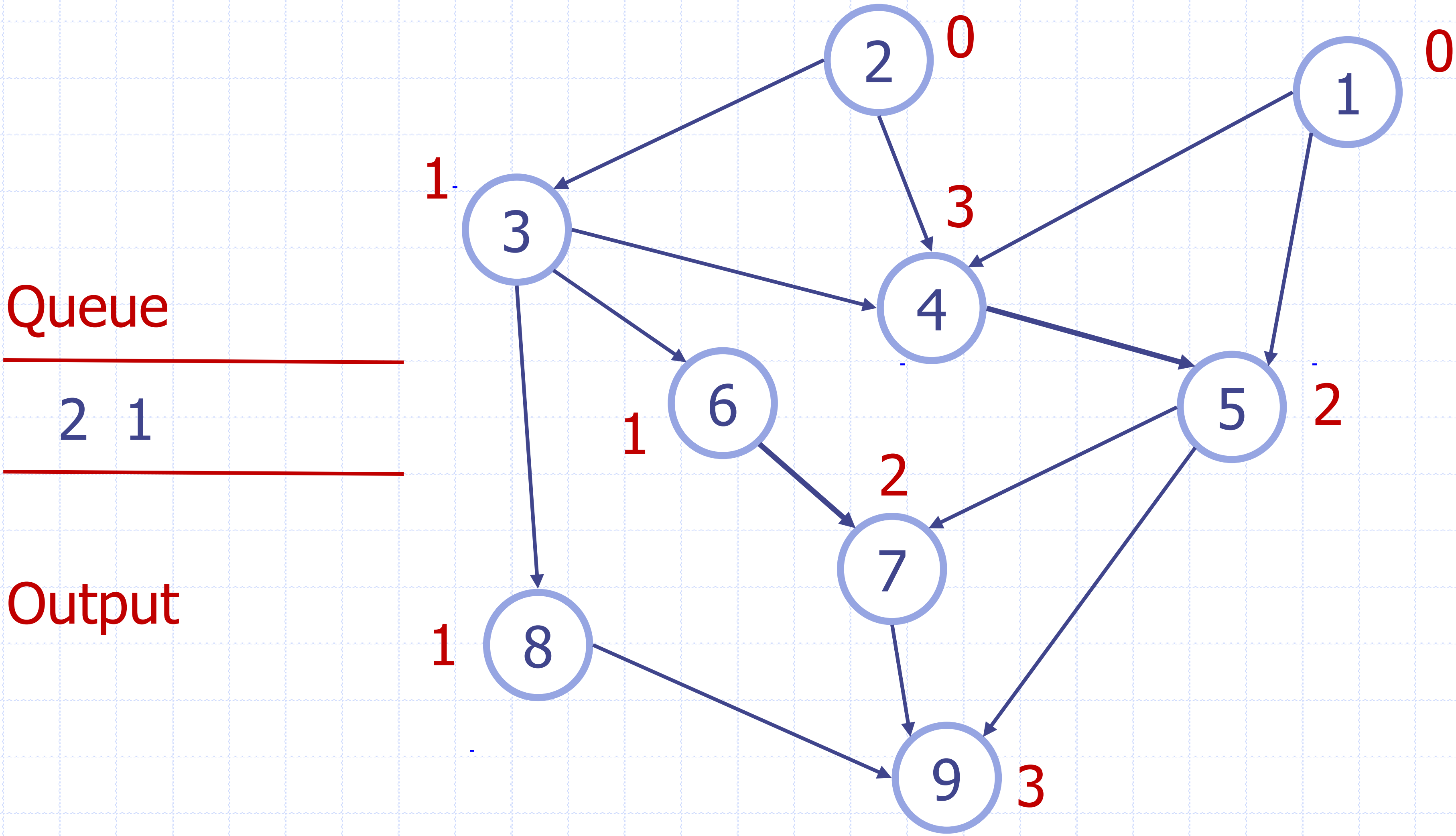
Compute in-degrees



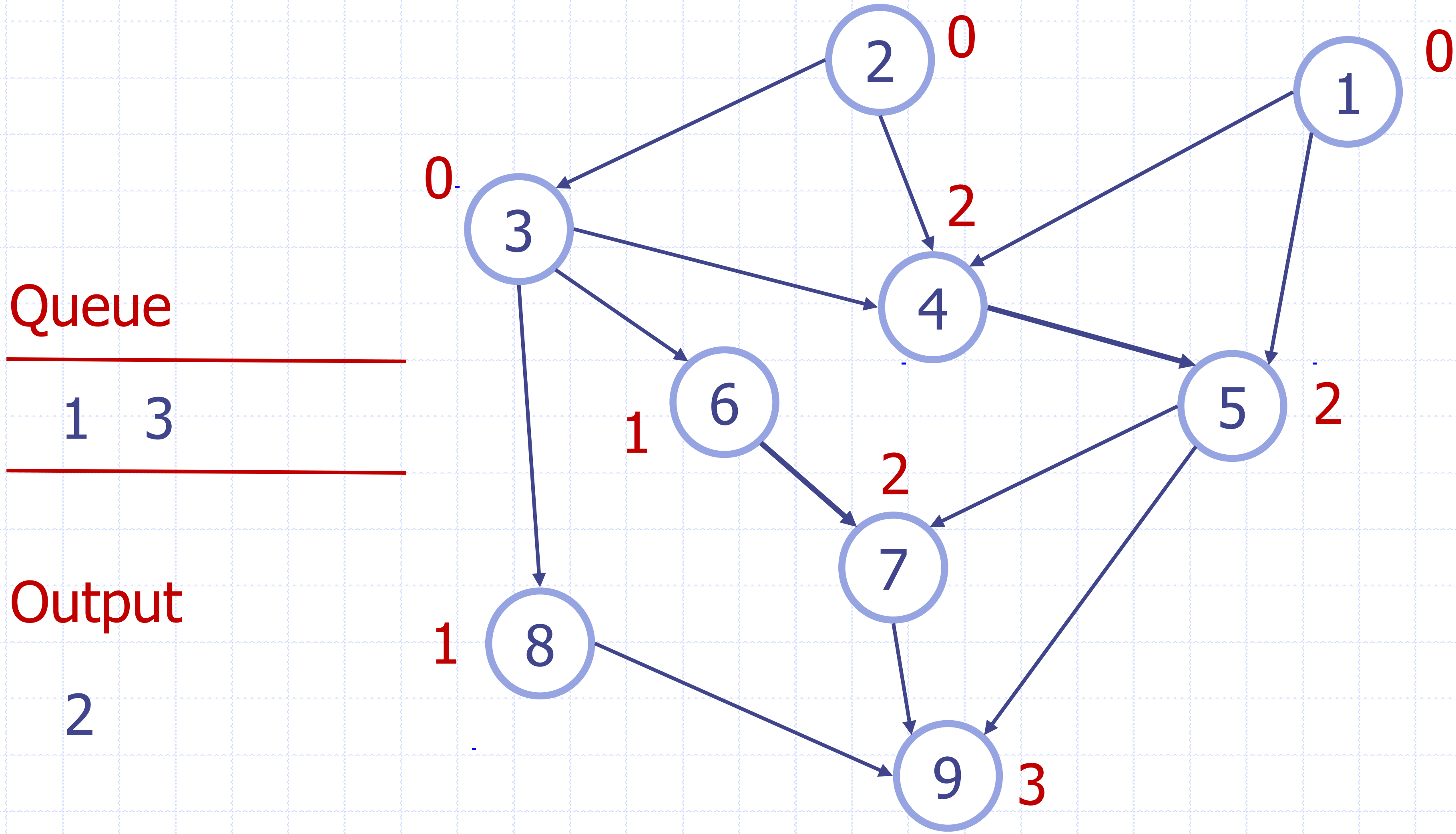
Queue

Output

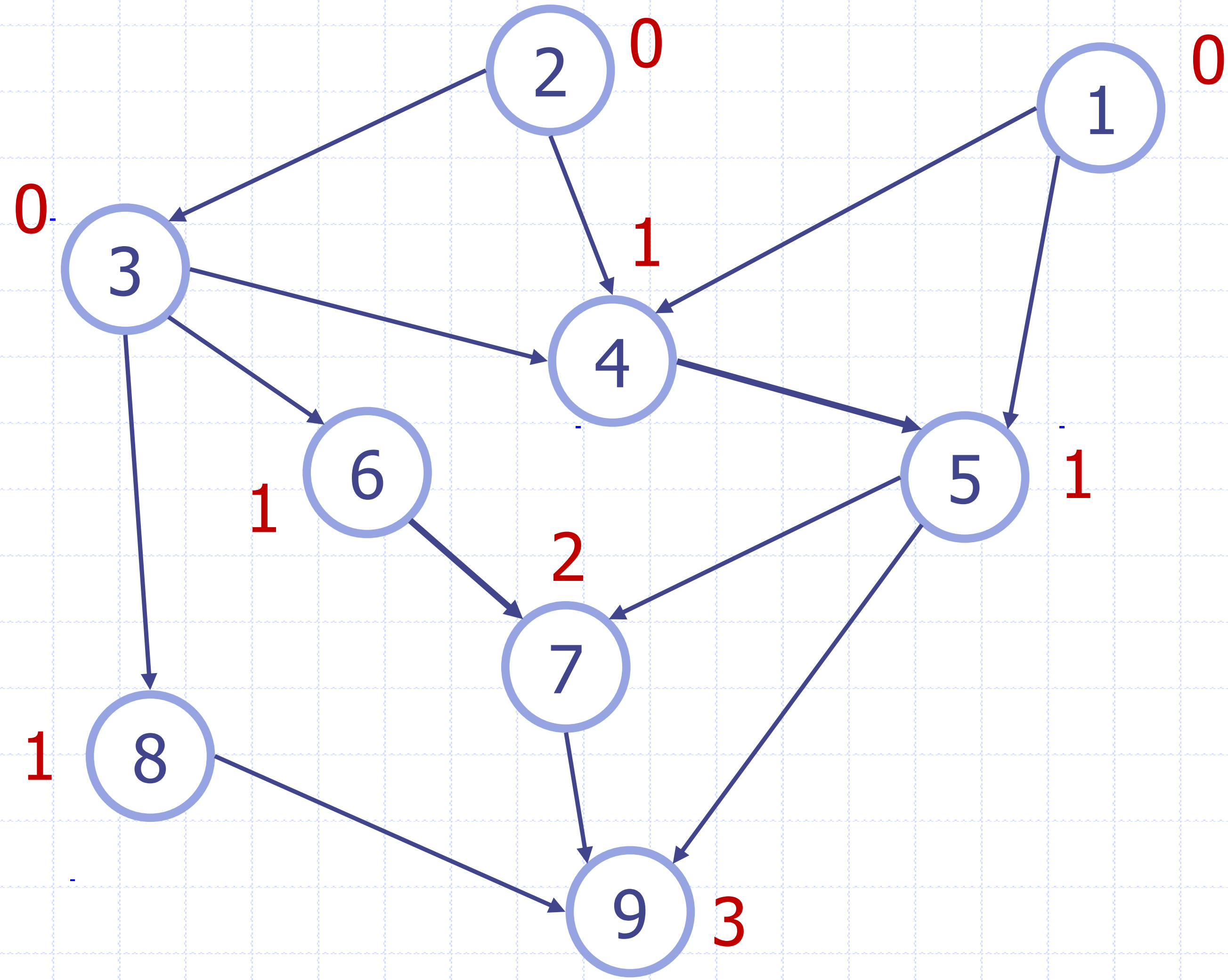
Topological Sorting Example



Topological Sorting Example



Topological Sorting Example



Queue

3

Output

2 1

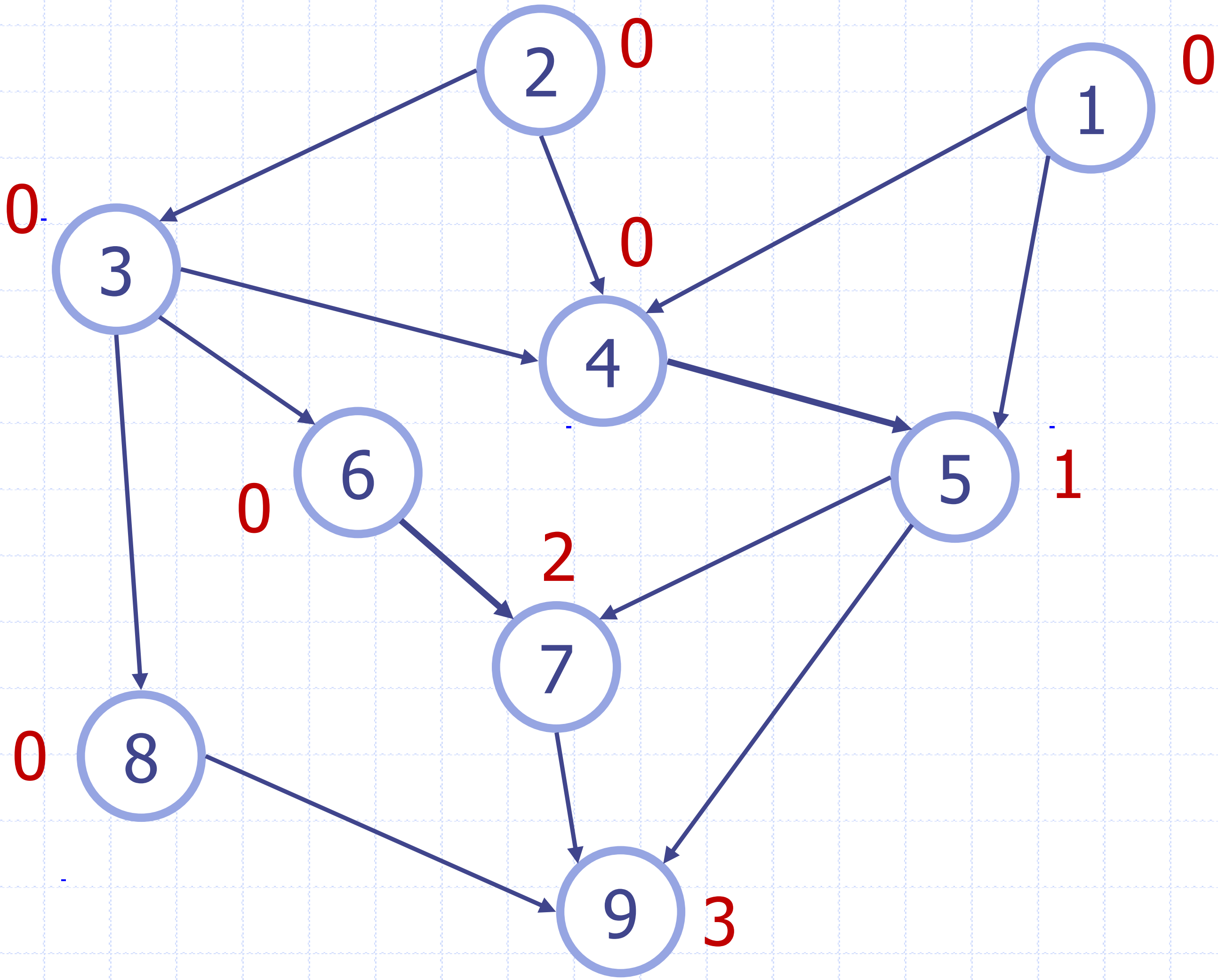
Topological Sorting Example

Queue

4 6 8

Output

2 1 3



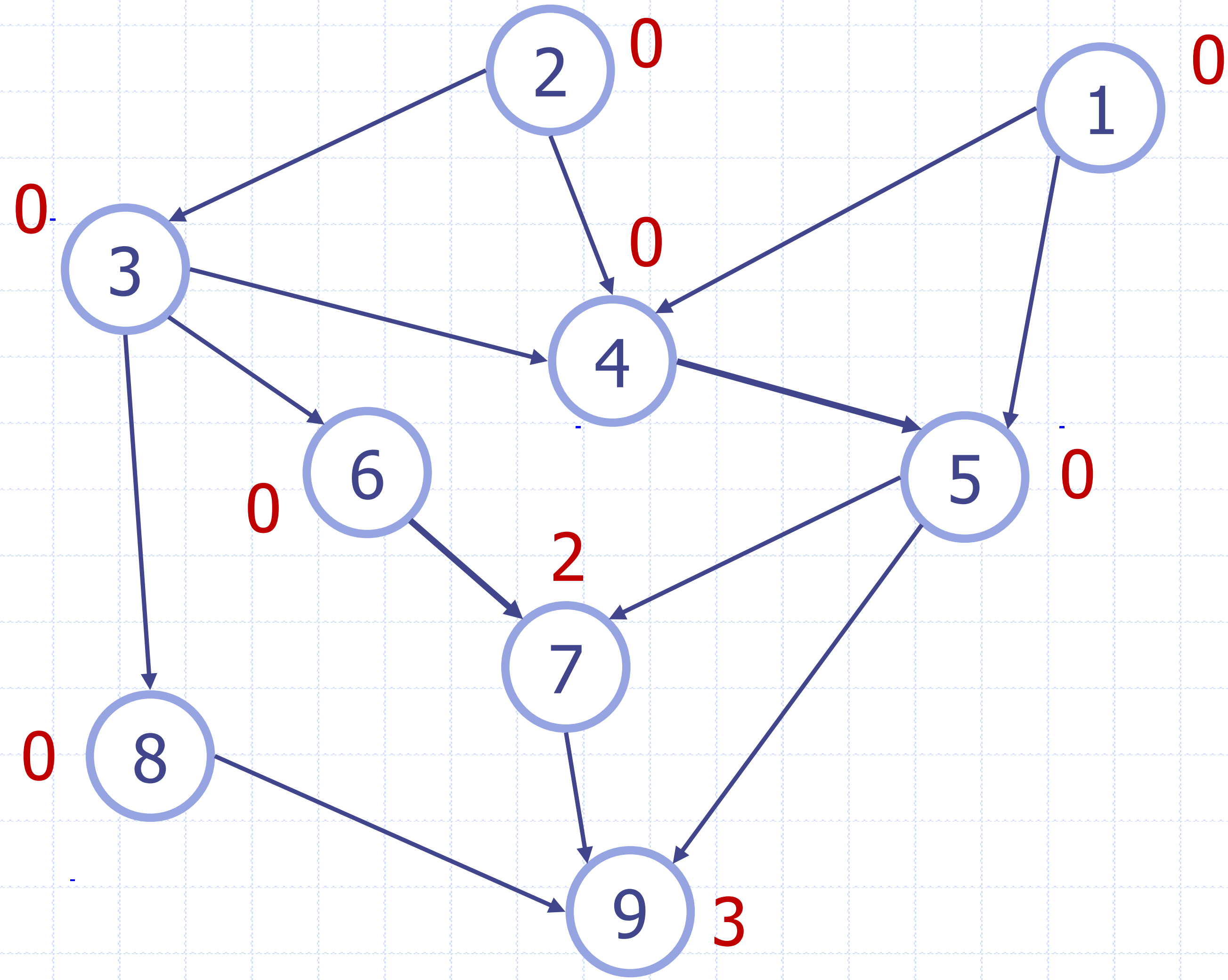
Topological Sorting Example

Queue

6 8 5

Output

2 1 3 4



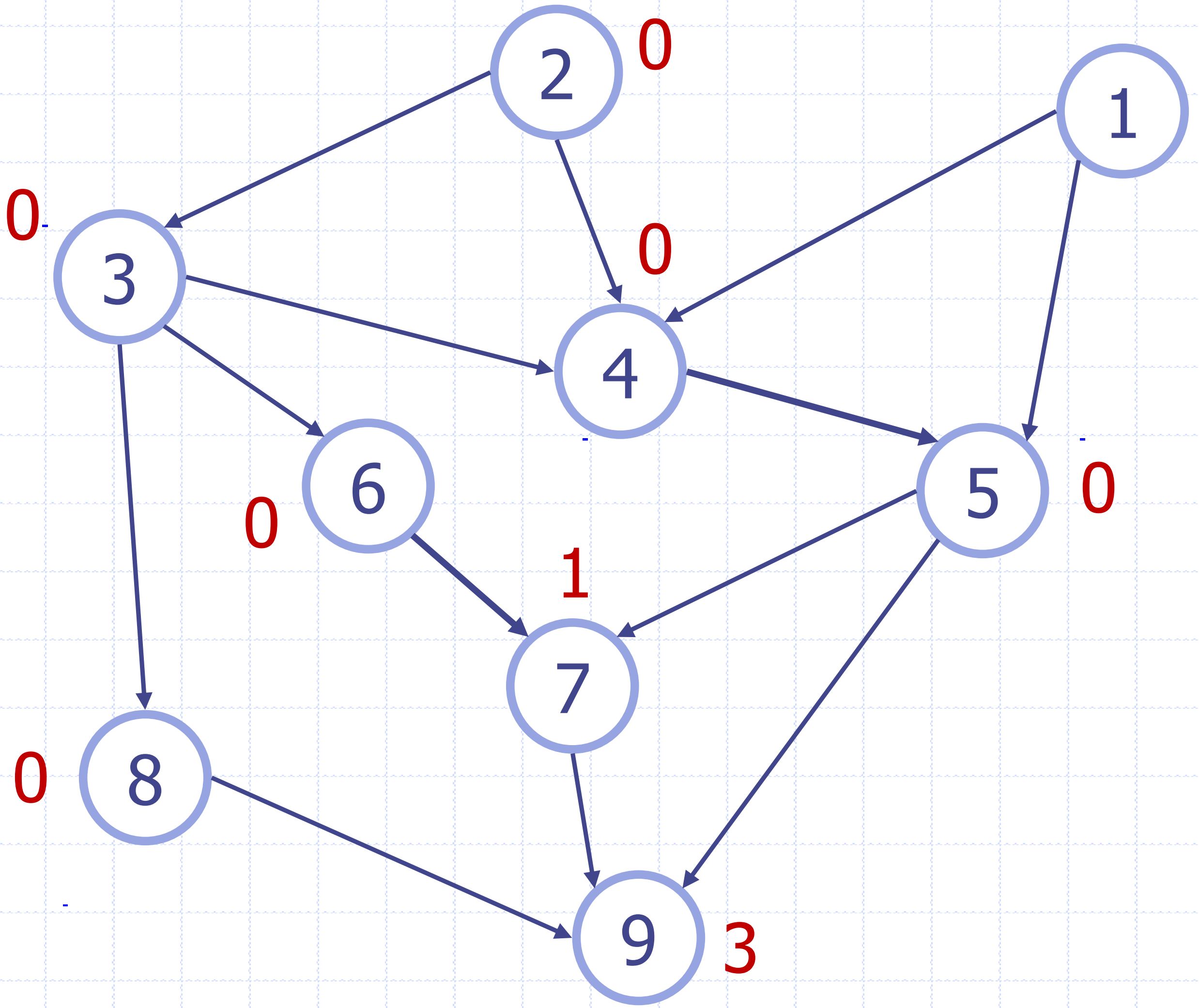
Topological Sorting Example

Queue

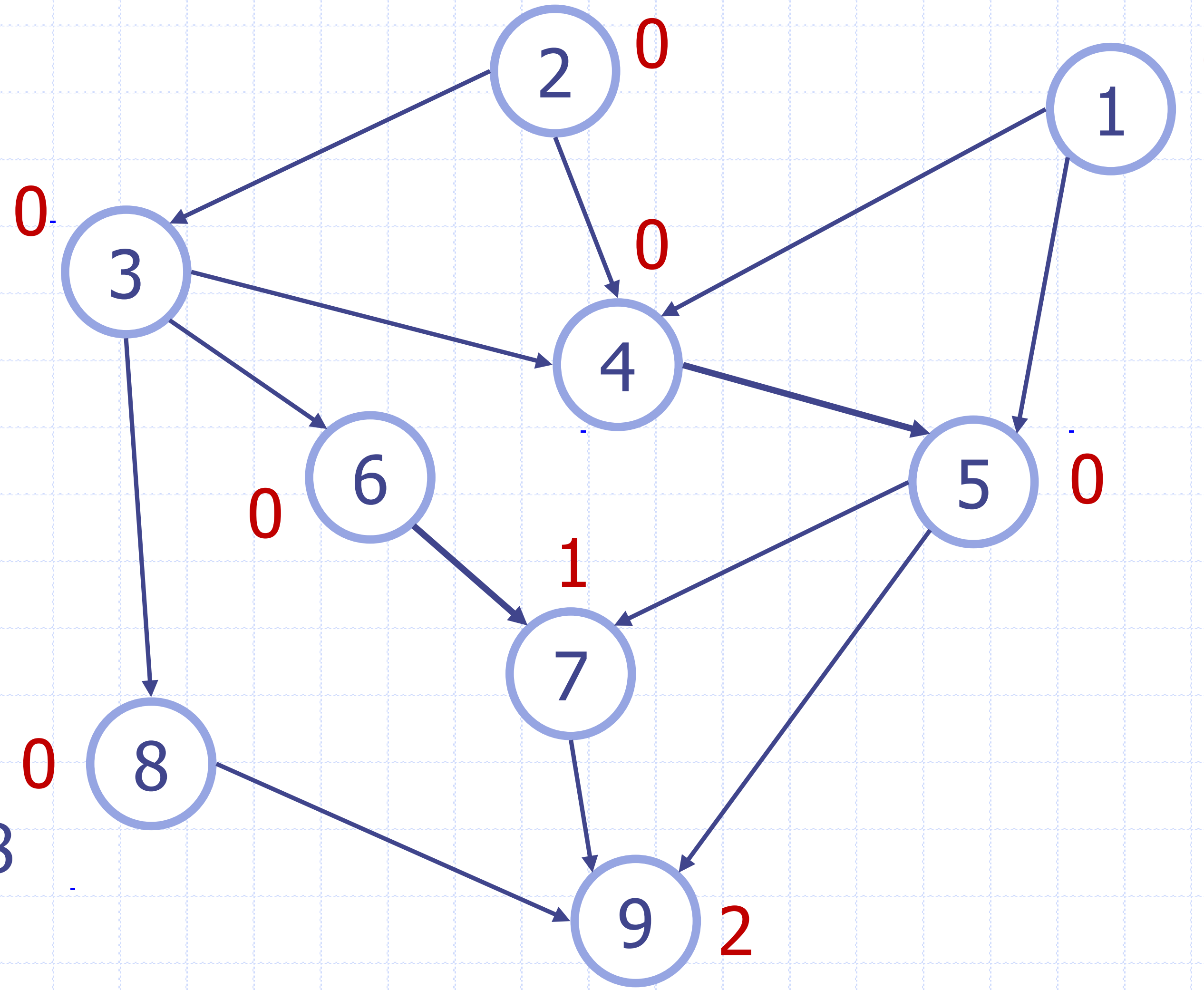
8 5

Output

2 1 3 4 6



Topological Sorting Example



Queue

5

Output

2 1 3 4 6 8

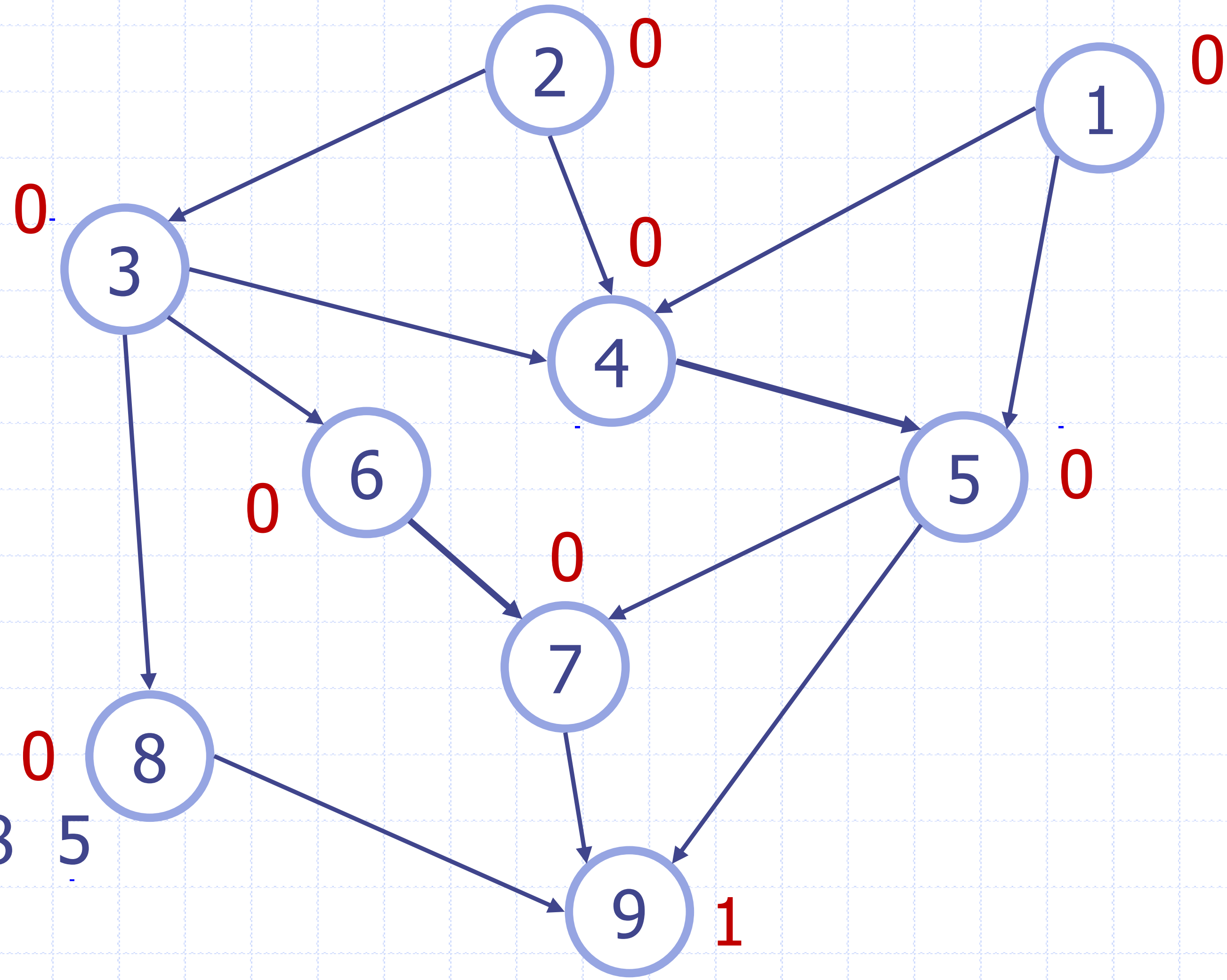
Topological Sorting Example

Queue

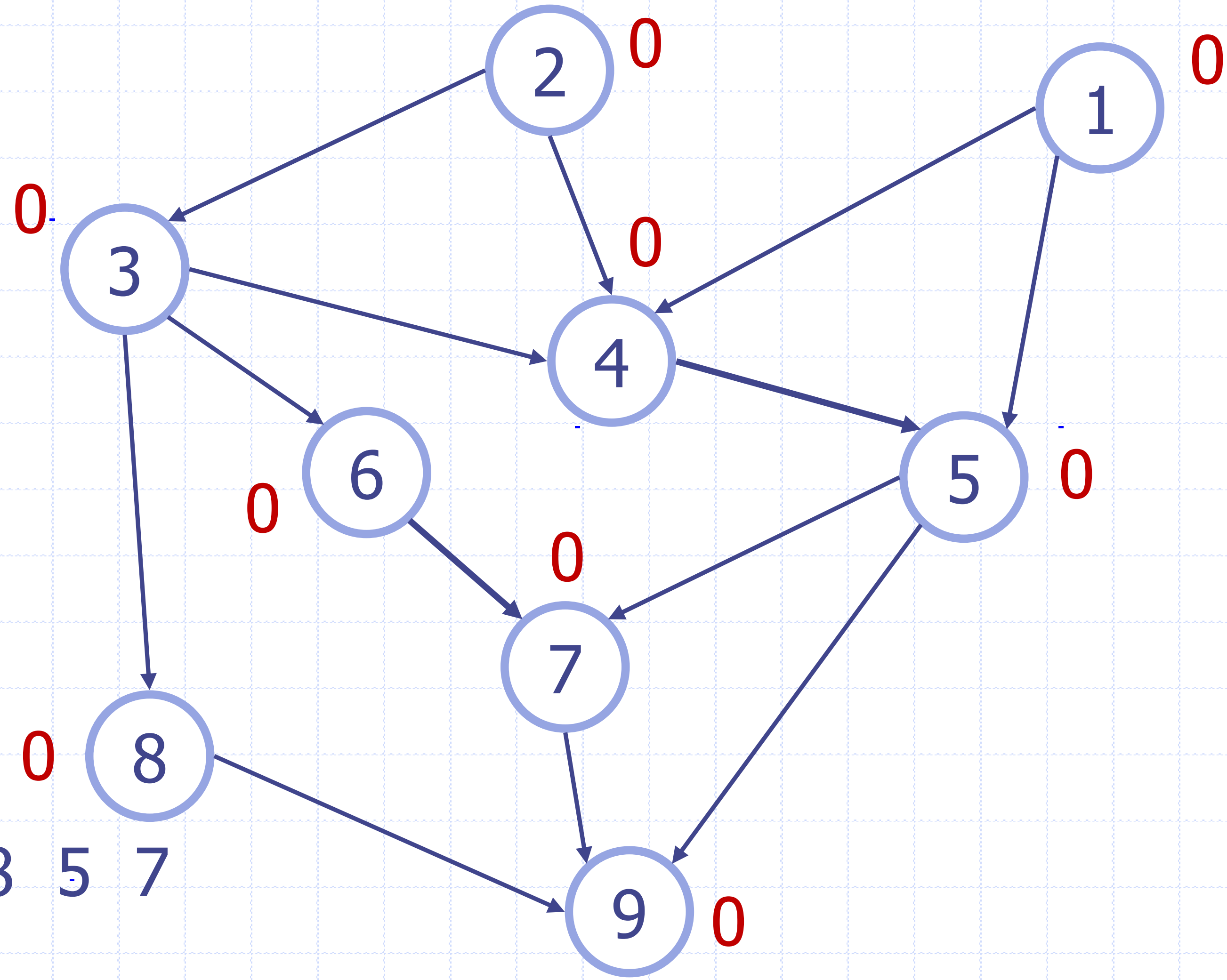
7

Output

2 1 3 4 6 8 5



Topological Sorting Example



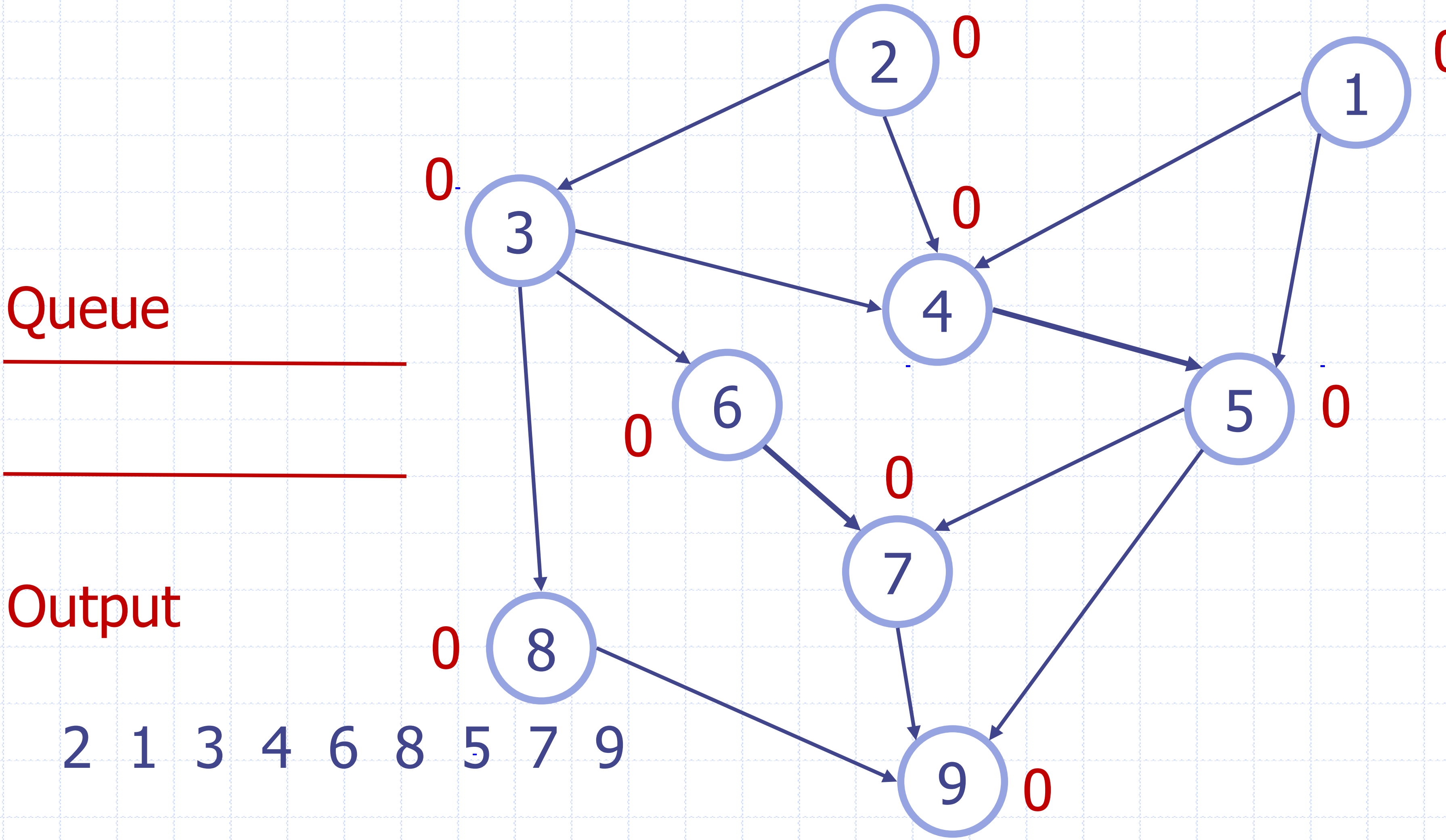
Queue

9

Output

2 1 3 4 6 8 5 7

Topological Sorting Example



2 1 3 4 6 8 5 7 9