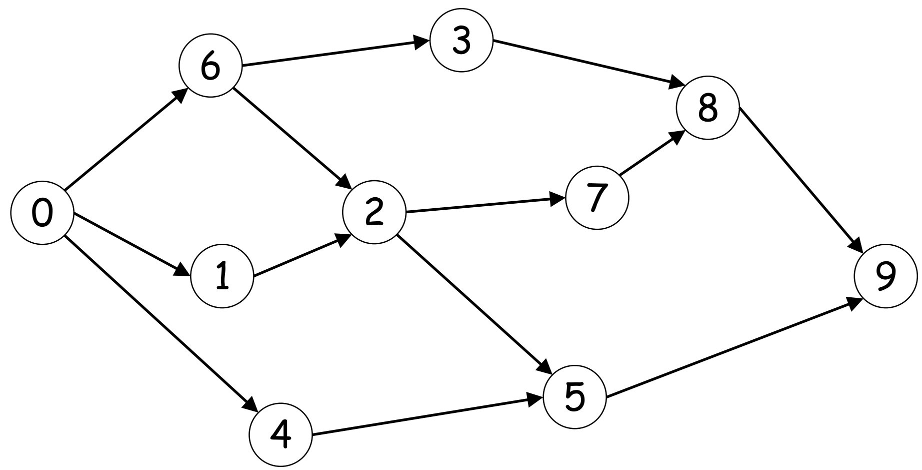
Topological Sort

Lecture 15

Directed Graph

A directed graph distinguishes between edge (u, v) and edge (v, u)

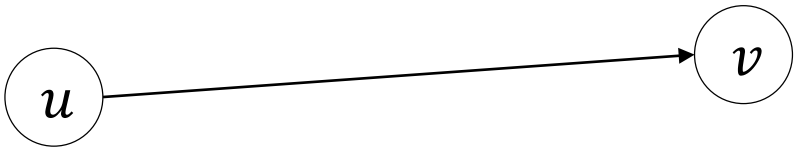


- out-degree of vertex v is the number of edges leaving v
- in-degree of vertex v is the number of edges entering v
- Each edge (u, v) contributes one to the out-degree of u and one to the in-degree of v, so

$$\sum_{v \in V} \text{out–degree}(v) = \sum_{v \in V} \text{in–degree}(v) = |E|$$

Directed Graphs

- Directed graphs are often used to represent order-dependent tasks
  - That is, we are told that task v cannot start before task u finishes
- Edge (u, v) denotes that task v cannot start until task u is finished



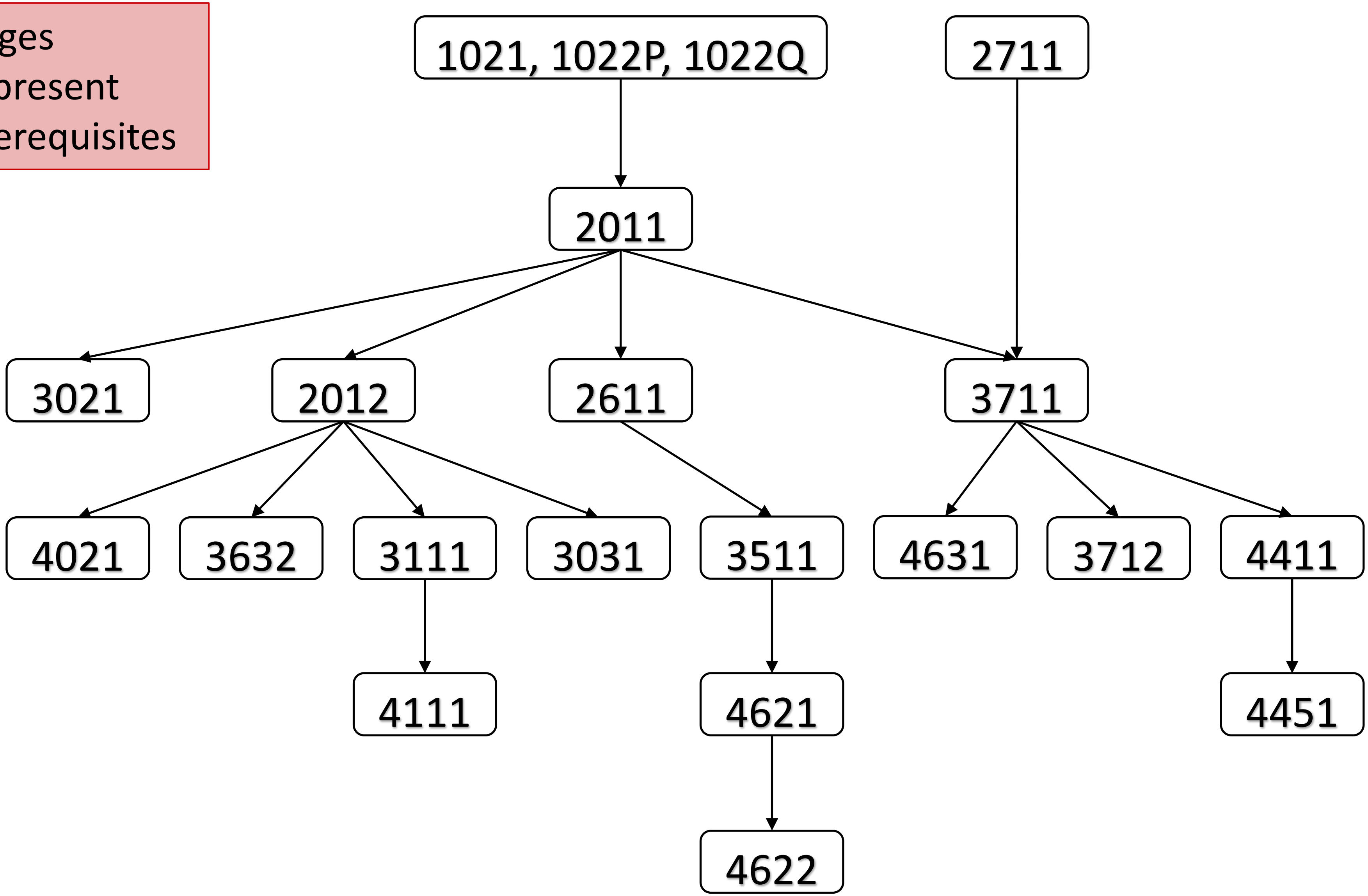
- Clearly, for a given set of tasks and order-dependencies, if the system doesn't hang, i.e., all tasks can be completed,

  the associated directed graph must be acyclic

  - Formally, the graph must be a Directed Acyclic Graph (DAG)

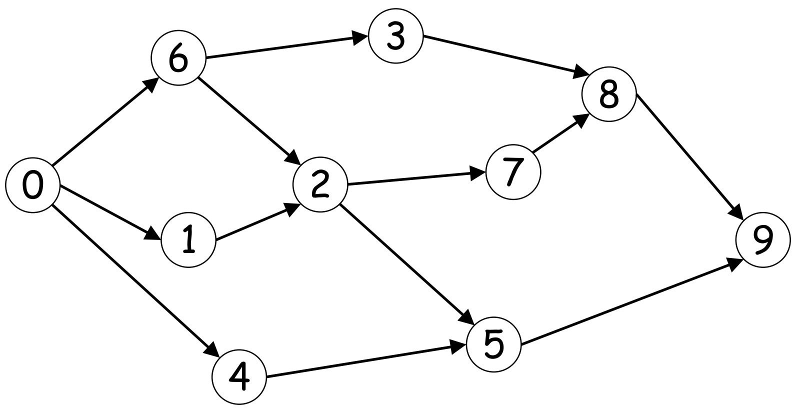
Partial COMP course dependency chart

Edges represent prerequisites



Topological Sort

- A Topological ordering of a graph is a linear ordering of the vertices of a DAG such that if (u, v) is in the graph, u appears before v in the linear ordering
- e.g., order in which classes can be taken



- Topological ordering may not be unique
- The graph above has many topological orderings
  - 0, 6, 1, 4, 3, 2, 5, 7, 8, 9
  - 0, 4, 1, 6, 2, 5, 3, 7, 8, 9
  - ...

Topological Sort Algorithm

- Observations
  - A DAG must contain at least one vertex with in-degree zero (why?)

- Algorithm: Topological Sort (TS)
  1. Output a vertex u with in-degree zero in current graph.
  2. Remove u and all edges (u, v) from current graph.
  3. If graph is not empty, goto step 1.

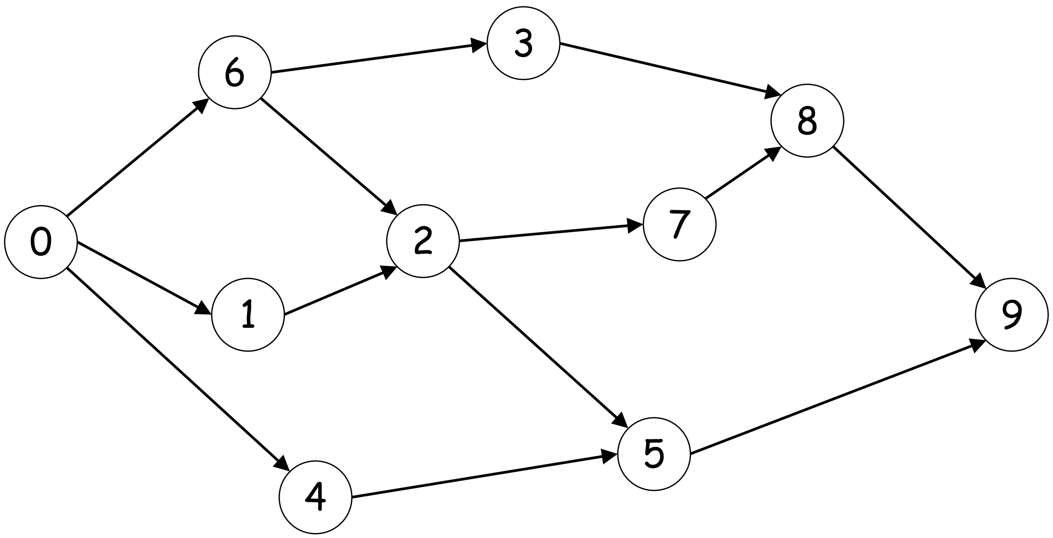
- Correctness
  - At every stage, current graph remains a DAG (why?)
  - Because current graph is always a DAG, TS can always output some vertex. So algorithm outputs all vertices.
  - Suppose outputted order was **not** a topological order.
    => Then there is some edge (u, v) such that v appears before u in the order. This is impossible, though, because v can not be output until edge (u, v) is removed!

Topological Sort Algorithm

Topological Sort(G)

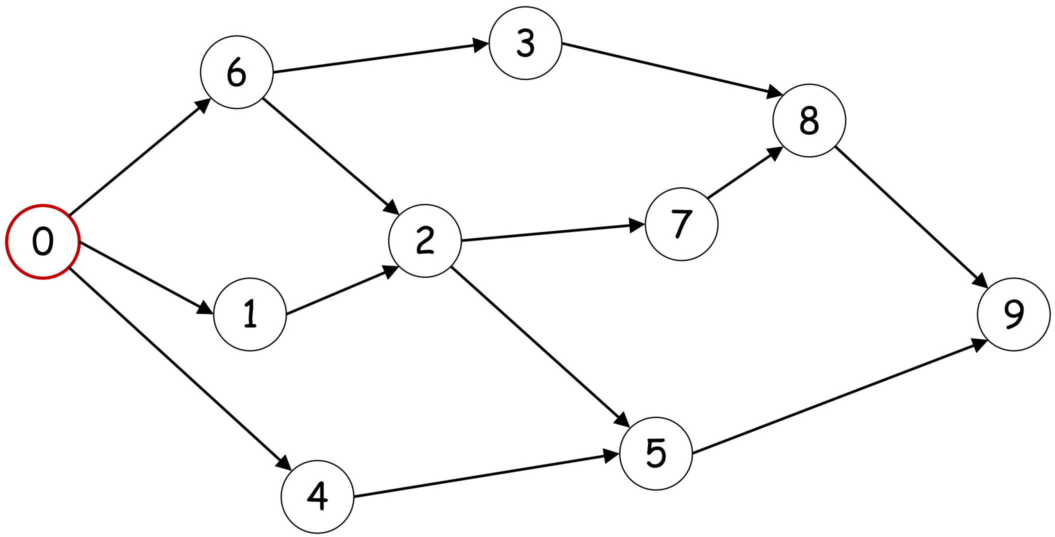
```
Initialize Q to be an empty queue;
foreach u in V do
    If in−degree(u) = 0 then
        // Find all starting vertices
        Enqueue(Q, v);
    end
end
while Q is not empty do
    u = Dequeue(Q);
    Output u;
    foreach v in Adj(u) do
        // remove u's outgoing edges
        in−degree(v) = in−degree(v) − 1
        if in−degree(v) = 0 then
            Enqueue(Q, v);
        end
    end
end
```

Example



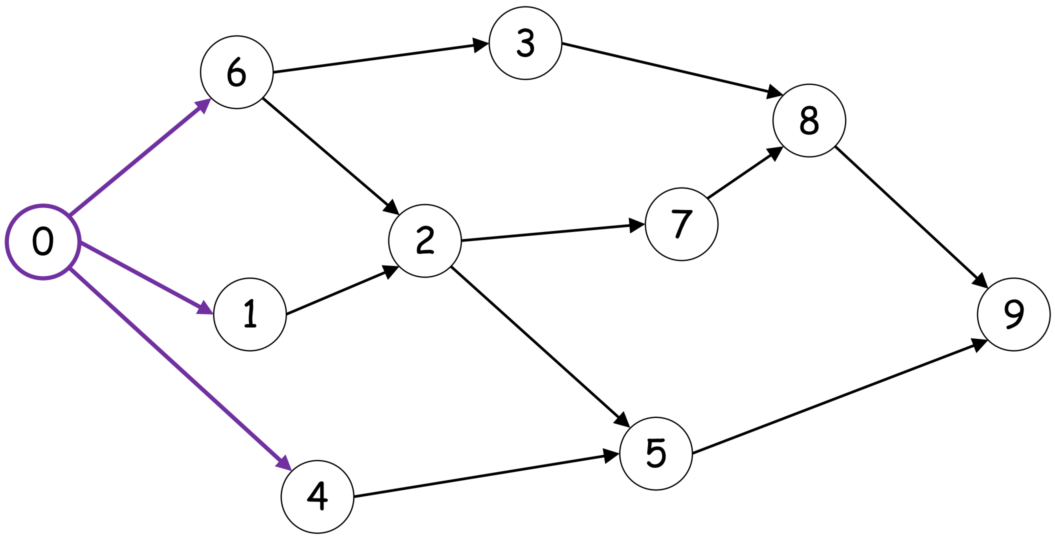
$Q = \{\}$

Example



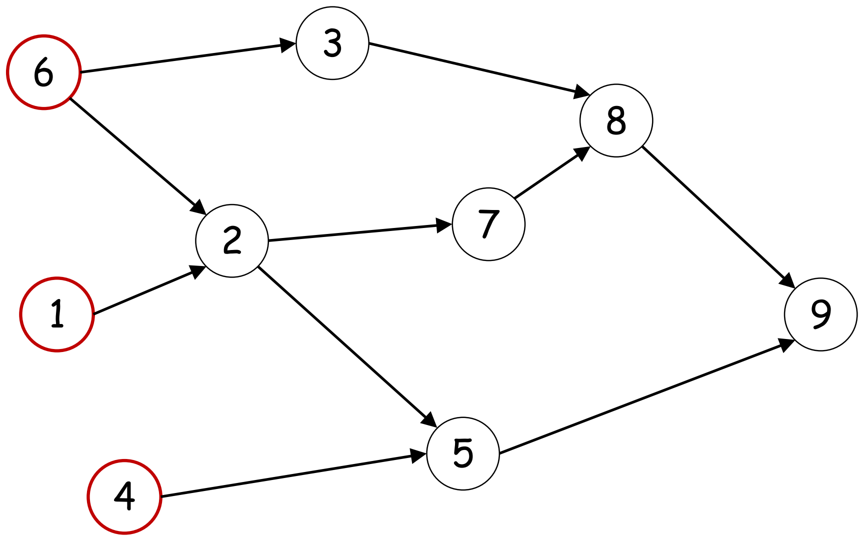
$Q = \{0\}$

Example



$Q = \{0\}$

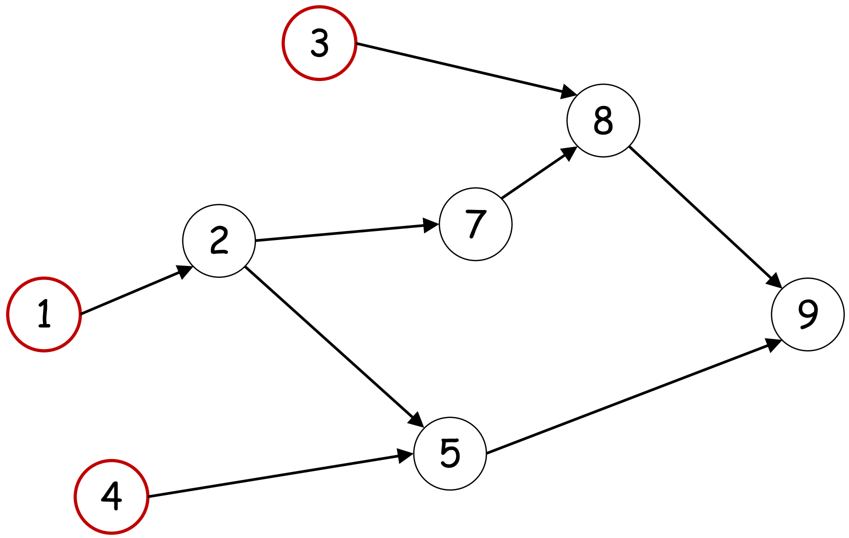
Example



$Q = \{6,1,4\}$

Output: 0

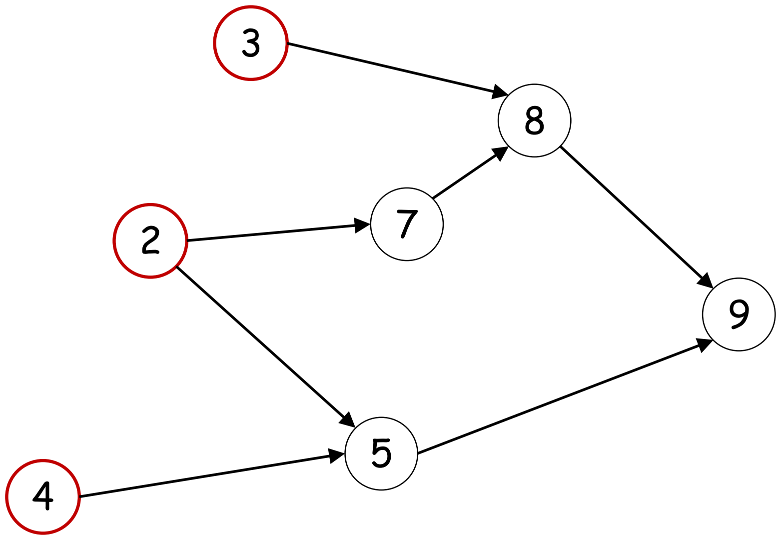
Example



$Q = \{1,4,3\}$

Output: 0,6

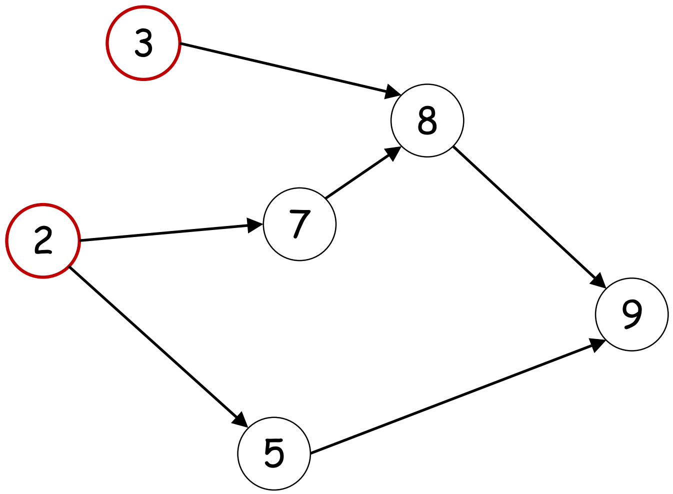
Example



$Q = \{4,3,2\}$

Output: 0,6,1

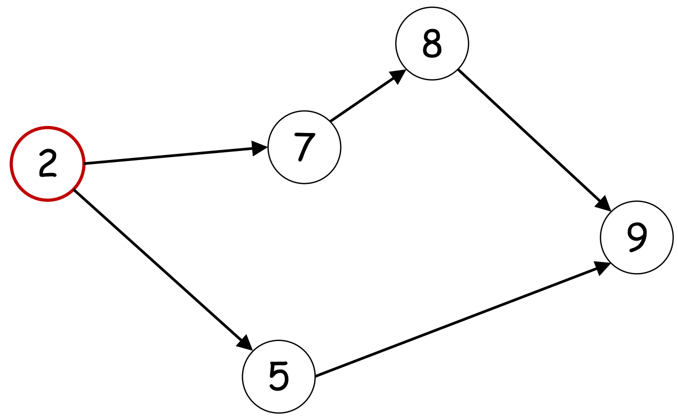
Example



$Q = \{3,2\}$

Output: 0,6,1,4

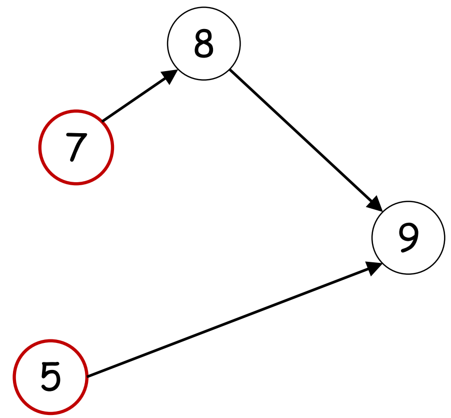
Example



$Q = \{2\}$

Output: 0,6,1,4,3

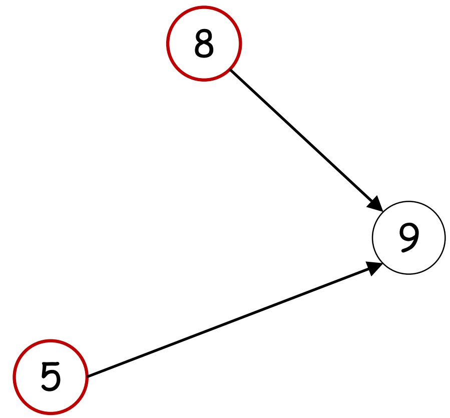
Example



$Q = \{7,5\}$

Output: 0,6,1,4,3,2

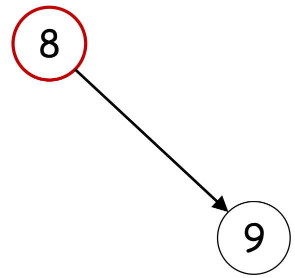
Example



$Q = \{5,8\}$

Output: 0,6,1,4,3,2,7

Example



$Q = \{8\}$

Output: 0,6,1,4,3,2,7,5

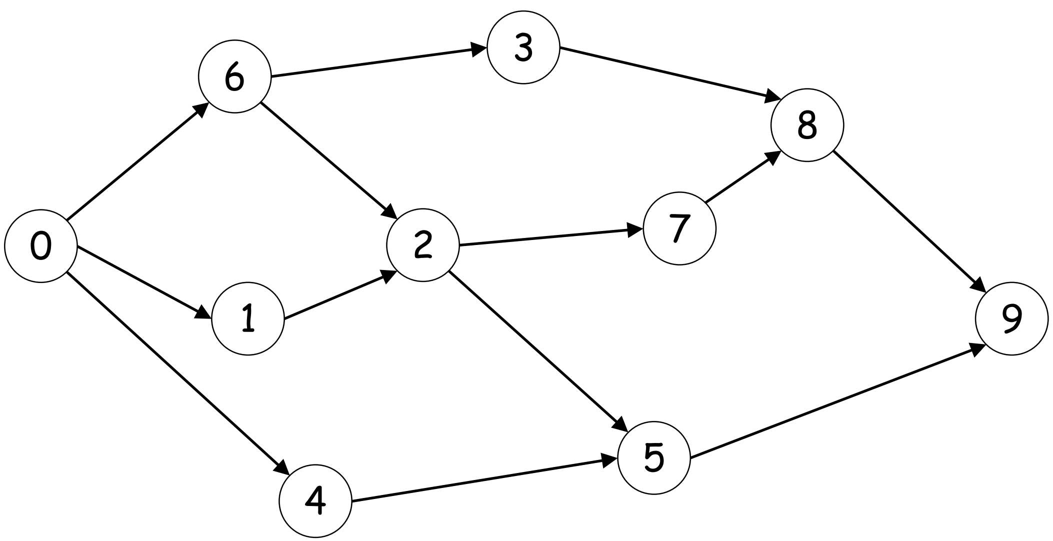
Example

9

$Q = \{9\}$

Output: 0,6,1,4,3,2,7,5,8

Example



$Q = \{\}$

Output: 0,6,1,4,3,2,7,5,8,9

Done!

Topological Sort: Complexity

- We never visit a vertex more than once
- For each vertex, we examine all outgoing edges
  - $\sum_{v \in V} \text{out}-\text{degree}(v) = E$
- Therefore, the running time is $O(V + E)$

Question

Can we use DFS to implement topological sort?