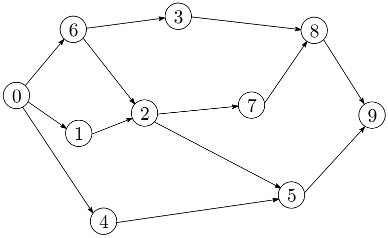
Topological Sort

Version of September 23, 2016

THE DEPARTMENT OF COMPUTER SCIENCE
計算機科學系 HKUST

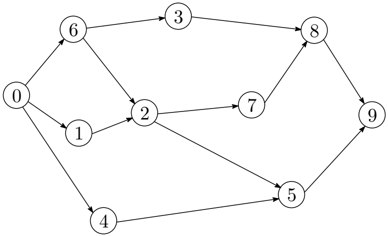
Directed Graph

In a directed graph, we distinguish between edge (u, v) and edge (v, u)



Directed Graph

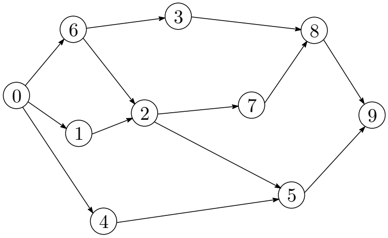
In a directed graph, we distinguish between edge (u, v) and edge (v, u)



- out-degree of a vertex is the number of edges leaving it

Directed Graph

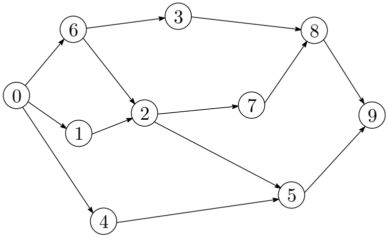
In a directed graph, we distinguish between edge (u, v) and edge (v, u)



- out-degree of a vertex is the number of edges leaving it
- in-degree of a vertex is the number of edges entering it

Directed Graph

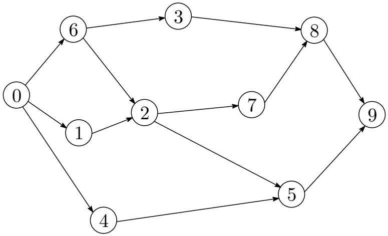
In a directed graph, we distinguish between edge (u, v) and edge (v, u)



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

Directed Graph

In a directed graph, we distinguish between edge (u, v) and edge (v, u)



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

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

- Directed graphs are often used to represent order-dependent tasks

Usage of Directed Graph

- Directed graphs are often used to represent order-dependent tasks
  - That is, we cannot start a task before another task finishes

Usage of Directed Graph

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

Usage of Directed Graph

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

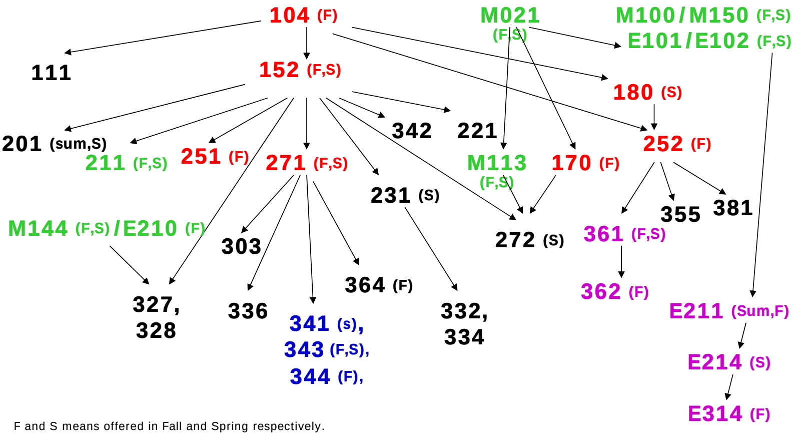
- Clearly, for the system not to hang, the graph must be acyclic

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

- Clearly, for the system not to hang, the graph must be acyclic
  - It must be a directed acyclic graph (or DAG)

Course dependence chart 09/10

Red: COMP/CSIE Core
Green: COMP/CSIE Required
Purple: CSIE (NW) Required
Blue: CSIE (MC) Required



F and S means offered in Fall and Spring respectively.
Course offering schedule shown here is for reference only;
the actual offering schedule may vary slightly from year to year.

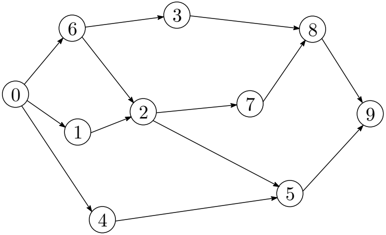
- A Topological ordering of a graph is a linear ordering of the vertices of a DAG such that if (u, v) is in the graph, u appears before v in the linear ordering

Topological Sort

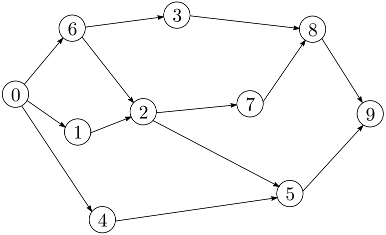
- A Topological ordering of a graph is a linear ordering of the vertices of a DAG such that if (u, v) is in the graph, u appears before v in the linear ordering
- e.g., order in which classes can be taken

Topological 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



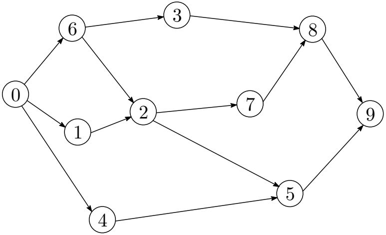
- A Topological ordering of a graph is a linear ordering of the vertices of a DAG such that if (u, v) is in the graph, u appears before v in the linear ordering
- e.g., order in which classes can be taken



- Topological ordering may not be unique as there are many "equal" elements!

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 as there are many "equal" elements!
- E.G., there are several 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?)

Topological Sort Algorithm

- Observations
  - A DAG must contain at least one vertex with in-degree zero (why?)
- Algorithm: Topological Sort
  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.

Topological Sort Algorithm

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

- Algorithm: Topological Sort
  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

Topological Sort Algorithm

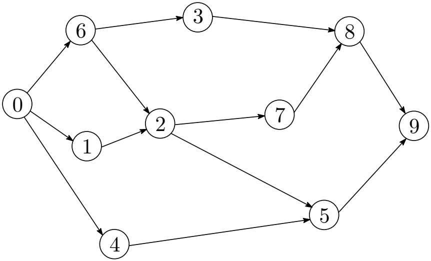
- Observations
  - A DAG must contain at least one vertex with in-degree zero (why?)
- Algorithm: Topological Sort
  1. Output a vertex u with in-degree zero in current graph.
  2. Remove u and all edges (u, v) from current graph.
  3. If graph is not empty, goto step 1.
- Correctness
  - At every stage, current graph is a DAG (why?)
  - Because current graph is always a DAG, algorithm can always output some vertex. So algorithm outputs all vertices.
  - Suppose order output 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

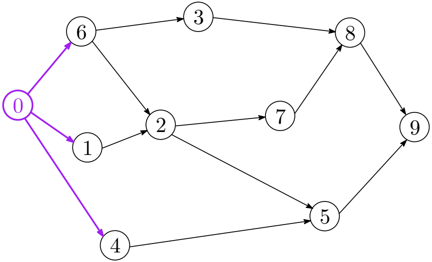
Topological_sort(G)

```
Initialize Q to be an empty queue;
foreach u in V do
    if in-degree(u) = 0 then
        // Find all starting vertices
        Enqueue(Q, u);
    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

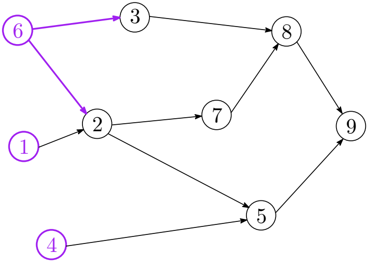


$Q = \{\}$



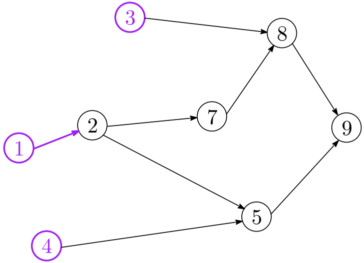
Q= {0}

Example



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

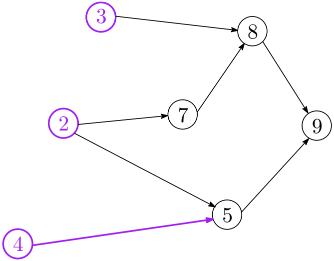
Output: 0



Q= {1, 4, 3}

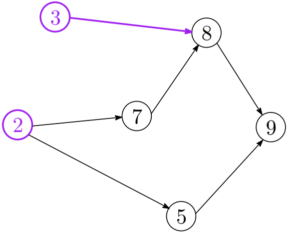
Output: 0, 6

Example



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

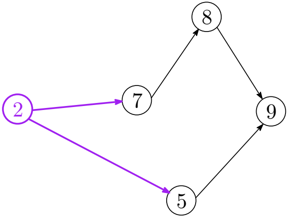
Output: 0, 6, 1



$Q = \{3, 2\}$

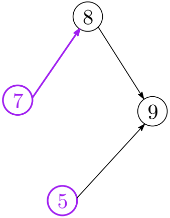
Output: 0, 6, 1, 4

Example



$Q = \{2\}$

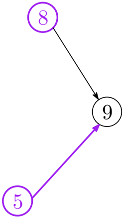
Output: 0, 6, 1, 4, 3



$Q = \{7, 5\}$

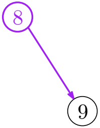
Output: 0, 6, 1, 4, 3, 2

Example



$Q = \{5, 8\}$

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

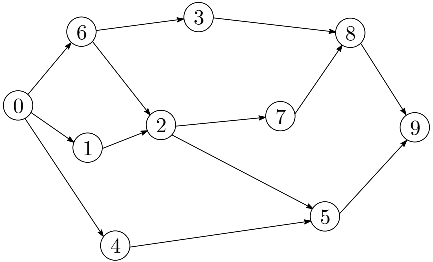


$Q = \{8\}$

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

$Q = \{9\}$

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



$Q = \{\}$

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

Done!

Topological Sort: Complexity

- We never visit a vertex more than once

Topological Sort: Complexity

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

Topological Sort: Complexity

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

Topological Sort: Complexity

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

Question

Can we use DFS to implement topological sort?