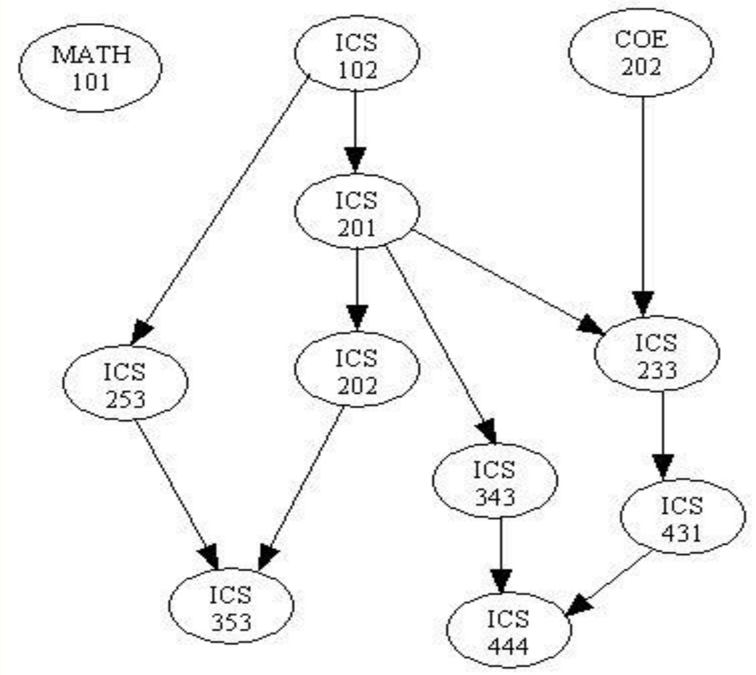
Topological Sorting

Kumkum Saxena

Introduction

- There are many problems involving a set of tasks in which some of the tasks must be done before others.
- For example, consider the problem of taking a course only after taking its prerequisites.
- Is there any systematic way of linearly arranging the courses in the order that they should be taken?

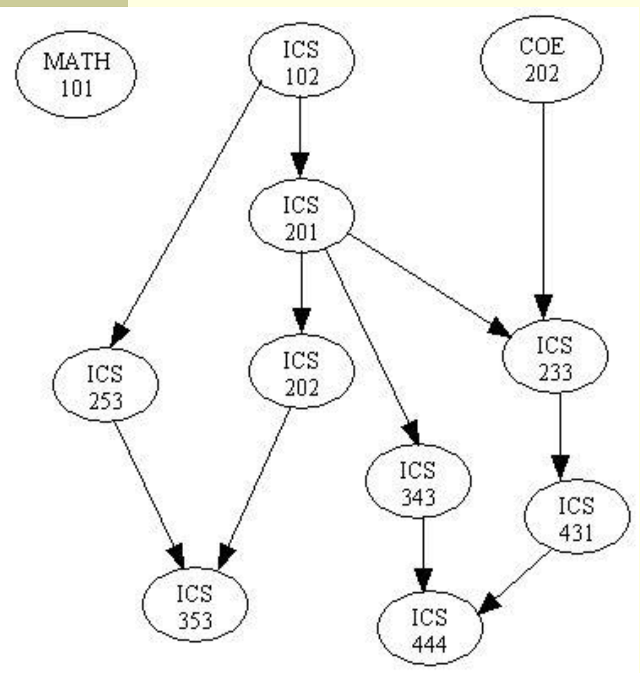


Yes! - Topological sort.

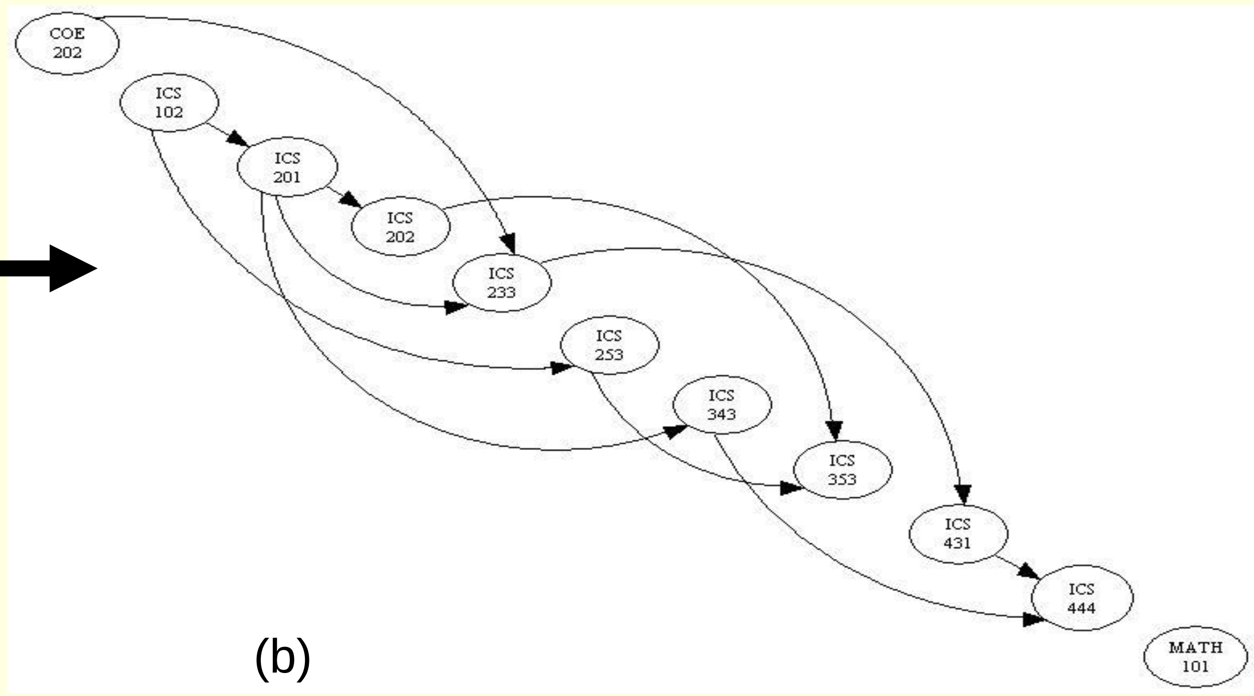
Definition of Topological Sort

- Topological sort is a method of arranging the vertices in a directed acyclic graph (DAG), as a sequence, such that no vertex appear in the sequence before its predecessor.

- The graph in (a) can be topologically sorted as in (b)



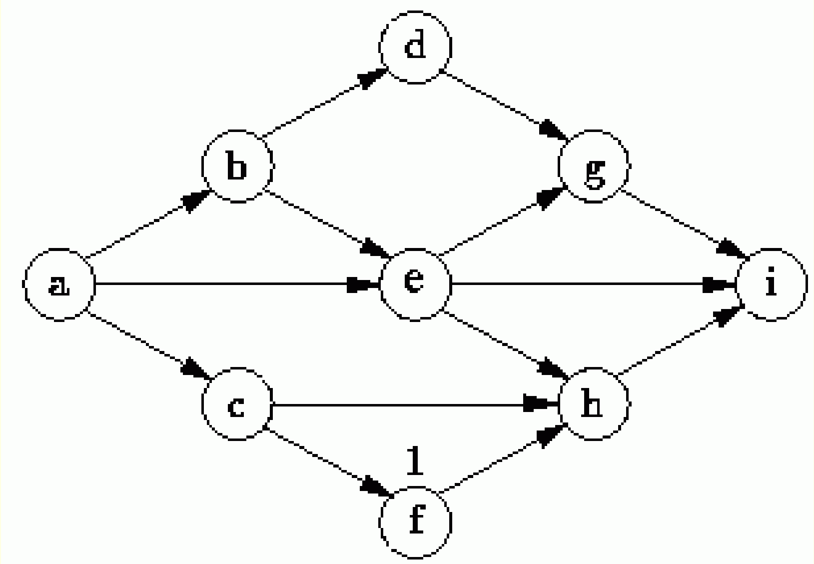
(a)



(b)

Topological Sort is not unique

- Topological sort is not unique.
- The following are all topological sort of the graph below:



s1 = {a, b, c, d, e, f, g, h, i}

s2 = {a, c, b, f, e, d, h, g, i}

s3 = {a, b, d, c, e, g, f, h, i}

s4 = {a, c, f, b, e, h, d, g, i}

etc.

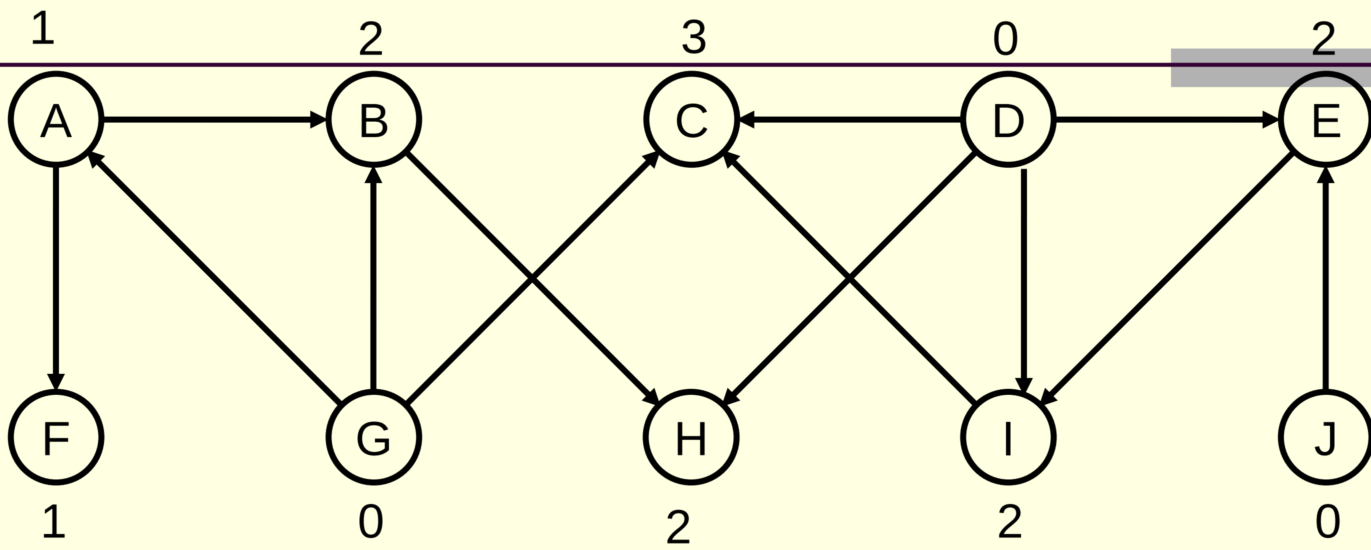
Topological Sort Algorithm

- One way to find a topological sort is to consider in-degrees of the vertices.
- The first vertex must have in-degree zero -- every DAG must have at least one vertex with in-degree zero.
- The Topological sort algorithm is:

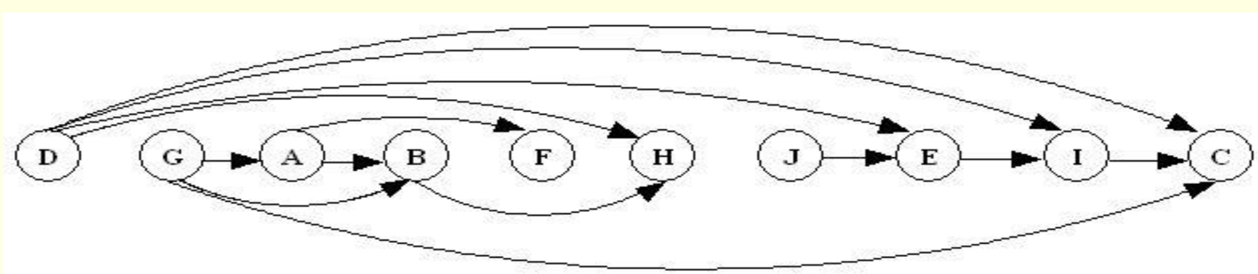
```
int topologicalOrderTraversal( ){
    int numVisitedVertices = 0;
    while(there are more vertices to be visited){
        if(there is no vertex with in-degree 0)
             break;
        else{
         select a vertex v that has in-degree 0;
         visit v;
         numVisitedVertices++;
         delete v and all its emanating edges;
       }
    }

   return numVisitedVertices;
}
```

Topological Sort Example

- **Demonstrating Topological Sort.**

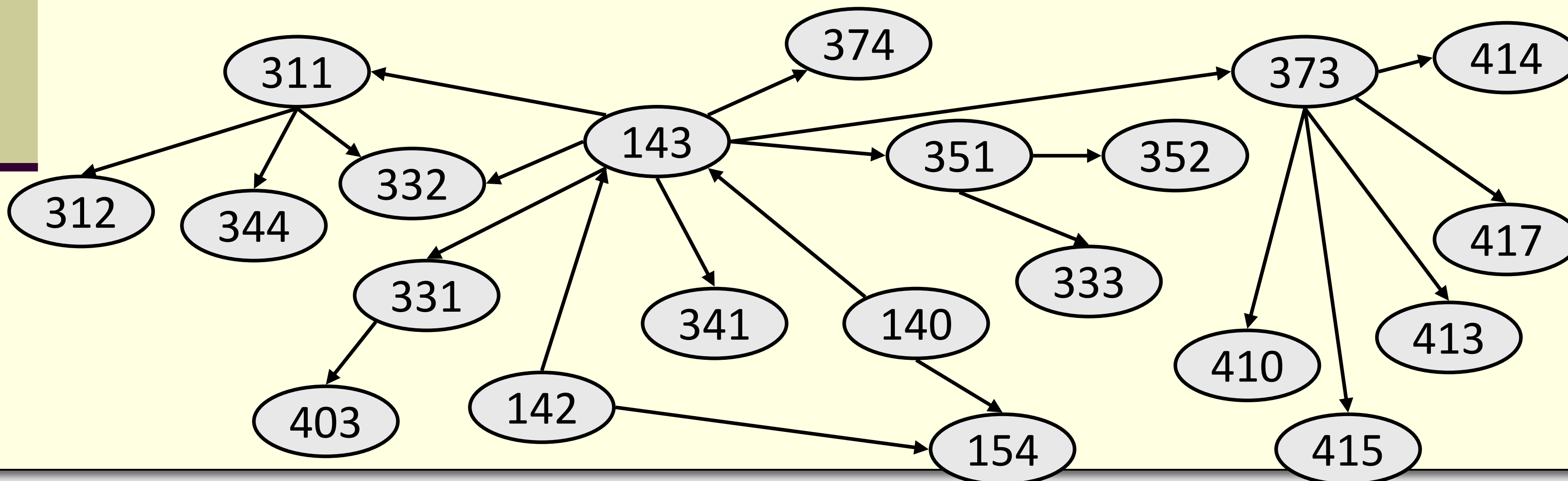


D	G	A	B	F	H	J	E	I	C



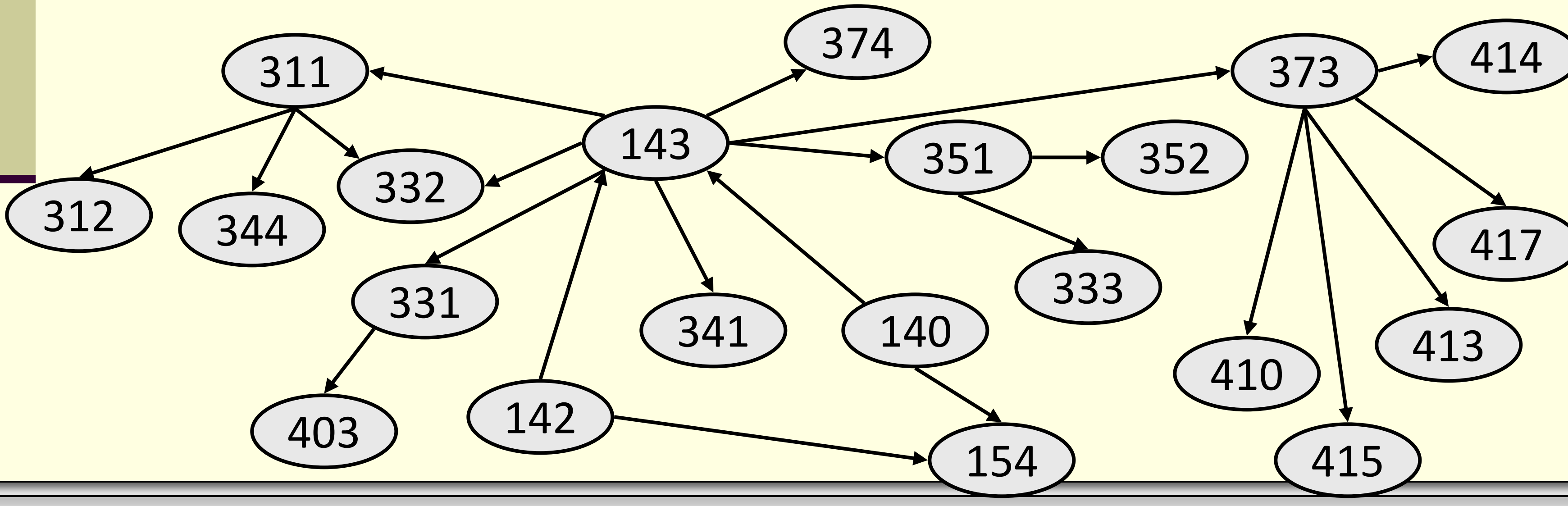
Ordering a graph

- Suppose we have a directed acyclic graph (DAG) of courses, and we want to find an order in which the courses can be taken.
  - Must take all prereqs before you can take a given course. Example:
    - [142, 143, 140, 154, 341, 374, 331, 403, 311, 332, 344, 312, 351, 333, 352, 373, 414, 410, 417, 413, 415]
    - There might be more than one allowable ordering.
  - How can we find a valid ordering of the vertices?



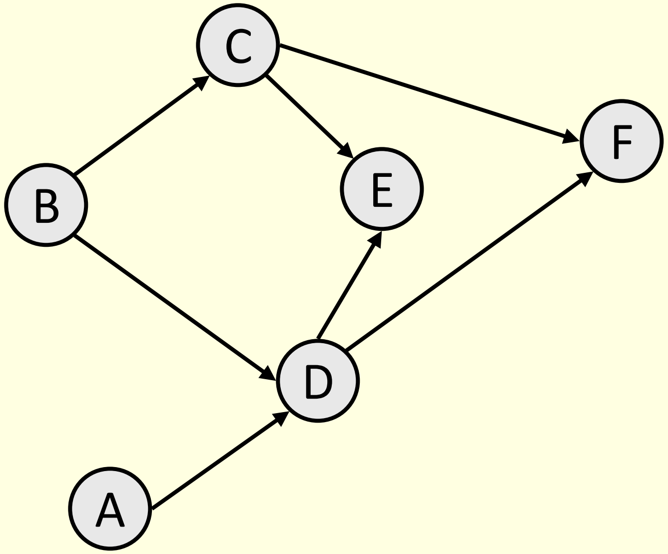
Topological Sort

- **topological sort:** Given a digraph $G = (V, E)$, a total ordering of G's vertices such that for every edge (v, w) in E, vertex v precedes w in the ordering. Examples:
  - determining the order to recalculate updated cells in a spreadsheet
  - finding an order to recompile files that have dependencies
    - (any problem of finding an order to perform tasks with dependencies)



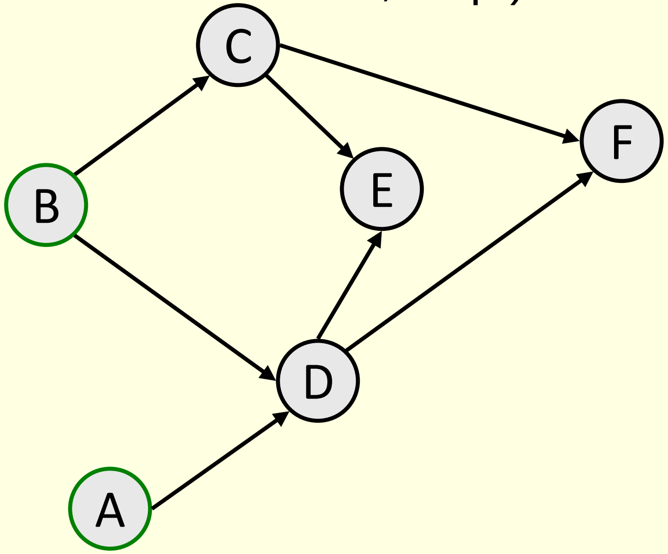
Topo sort example

- How many valid topological sort orderings can you find for the vertices in the graph below?
  - [A, B, C, D, E, F], [A, B, C, D, F, E],
  - [A, B, D, C, E, F], [A, B, D, C, F, E],
  - [B, A, C, D, E, F], [B, A, C, D, F, E],
  - [B, A, D, C, E, F], [B, A, D, C, F, E],
  - [B, C, A, D, E, F], [B, C, A, D, F, E],
  - ...



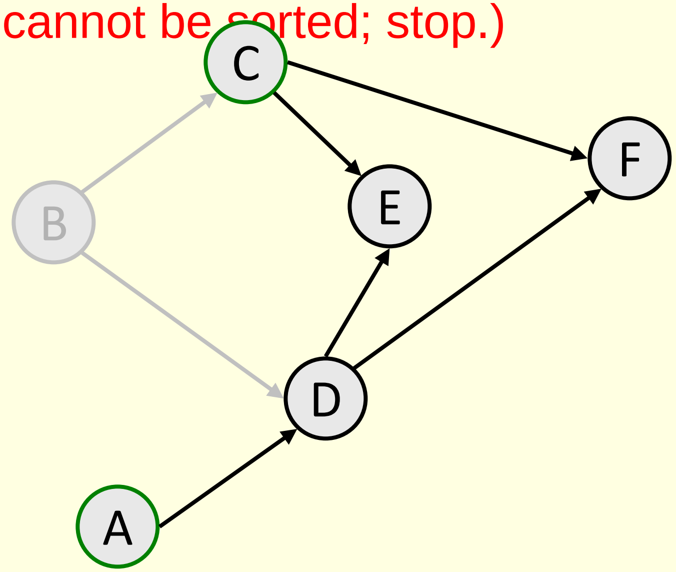
Topo sort: Algorithm 1

- function topologicalSort():
  - *ordering* := { }.
  - Repeat until graph is empty:
    - Find a vertex *v* with in-degree of 0 (no incoming edges).
      - (If there is no such vertex, the graph cannot be sorted; stop.)
    - Delete *v* and all of its outgoing edges from the graph.
    - *ordering* += *v* .



Topo sort example

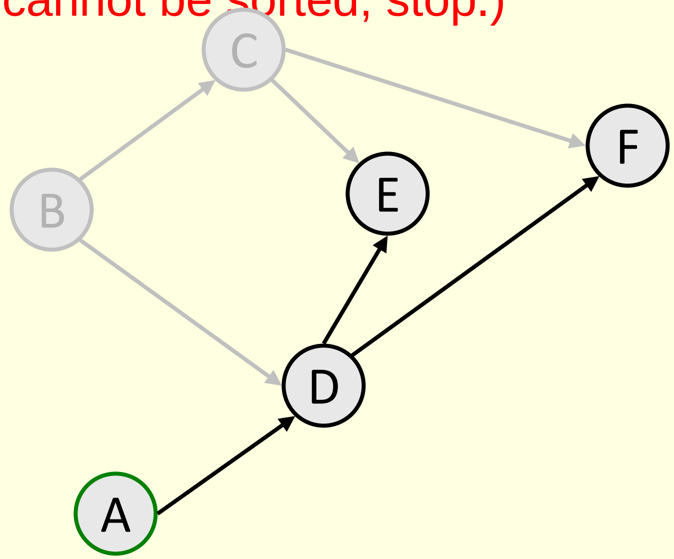
- function topologicalSort():
  - *ordering* := { }.
  - Repeat until graph is empty:
    - Find a vertex *v* with in-degree of 0 (no incoming edges).
      - (If there is no such vertex, the graph cannot be sorted; stop.)
    - Delete *v* and all of its outgoing edges from the graph.
    - *ordering* += *v* .



- *ordering* = { B }

Topo sort example

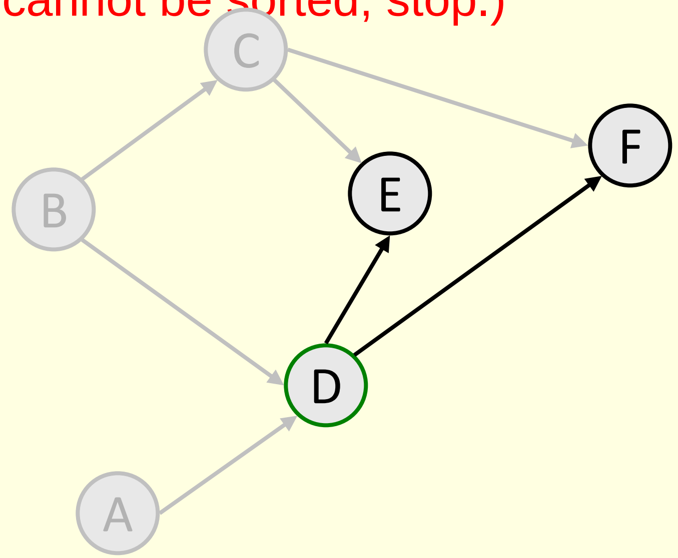
- function topologicalSort():
  - *ordering* := { }.
  - Repeat until graph is empty:
    - Find a vertex *v* with in-degree of 0 (no incoming edges).
      - (If there is no such vertex, the graph cannot be sorted; stop.)
    - Delete *v* and all of its outgoing edges from the graph.
    - *ordering* += *v* .



- *ordering* = { B, C }

Topo sort example

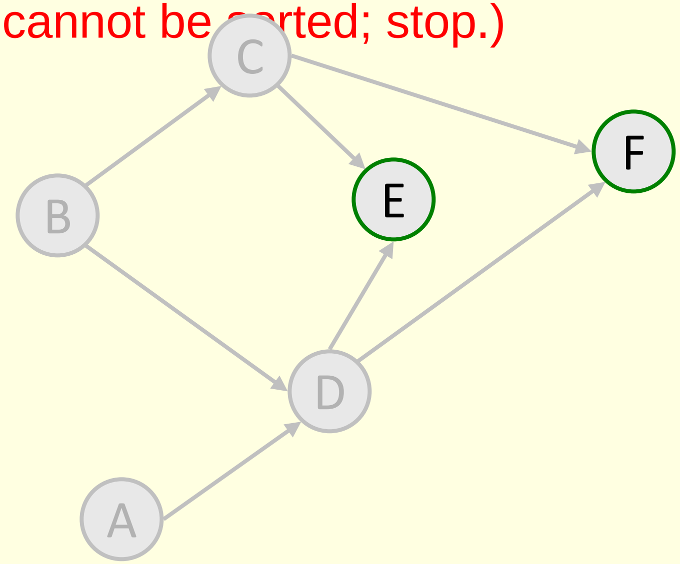
- function topologicalSort():
  - *ordering* := { }.
  - Repeat until graph is empty:
    - Find a vertex *v* with in-degree of 0 (no incoming edges).
      - (If there is no such vertex, the graph cannot be sorted; stop.)
    - Delete *v* and all of its outgoing edges from the graph.
    - *ordering* += *v* .



- *ordering* = { B, C, A }

Topo sort example

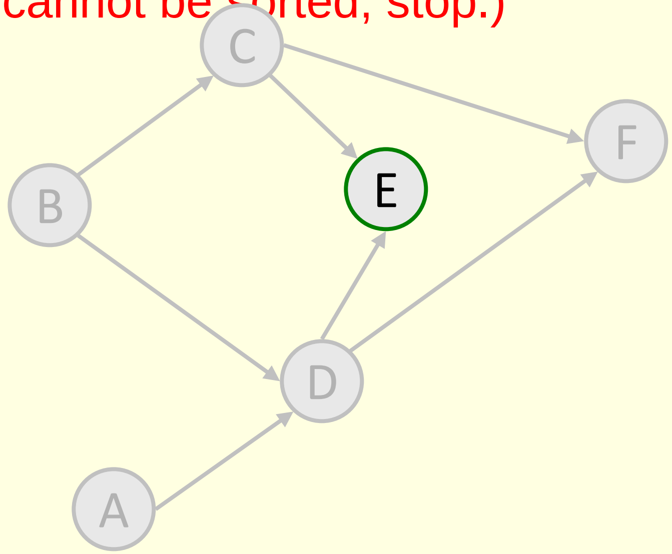
- function topologicalSort():
  - *ordering* := { }.
  - Repeat until graph is empty:
    - Find a vertex *v* with in-degree of 0 (no incoming edges).
      - (If there is no such vertex, the graph cannot be sorted; stop.)
    - Delete *v* and all of its outgoing edges from the graph.
    - *ordering* += *v* .



- *ordering* = { B, C, A, D }

Topo sort example

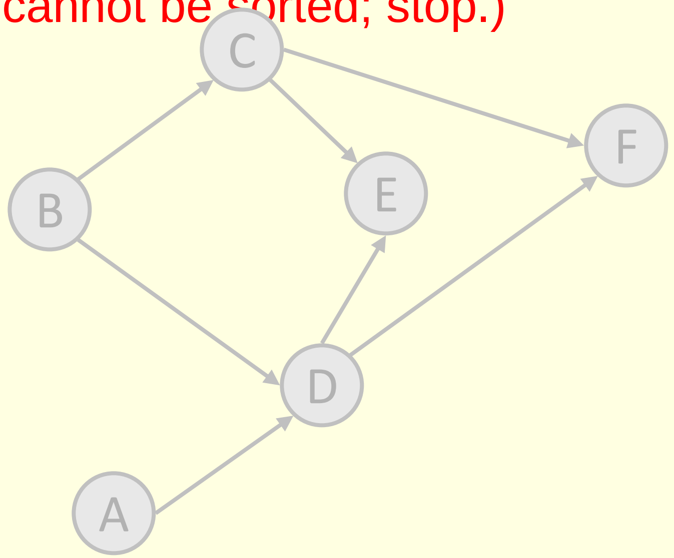
- function topologicalSort():
  - *ordering* := { }.
  - Repeat until graph is empty:
    - Find a vertex *v* with in-degree of 0 (no incoming edges).
      - (If there is no such vertex, the graph cannot be sorted; stop.)
    - Delete *v* and all of its outgoing edges from the graph.
    - *ordering* += *v* .



- *ordering* = { B, C, A, D, F }

Topo sort example

- function topologicalSort():
  - *ordering* := { }.
  - Repeat until graph is empty:
    - Find a vertex *v* with in-degree of 0 (no incoming edges).
      - (If there is no such vertex, the graph cannot be sorted; stop.)
    - Delete *v* and all of its outgoing edges from the graph.
    - *ordering* += *v* .
  - *ordering* = { B, C, A, D, F, E }

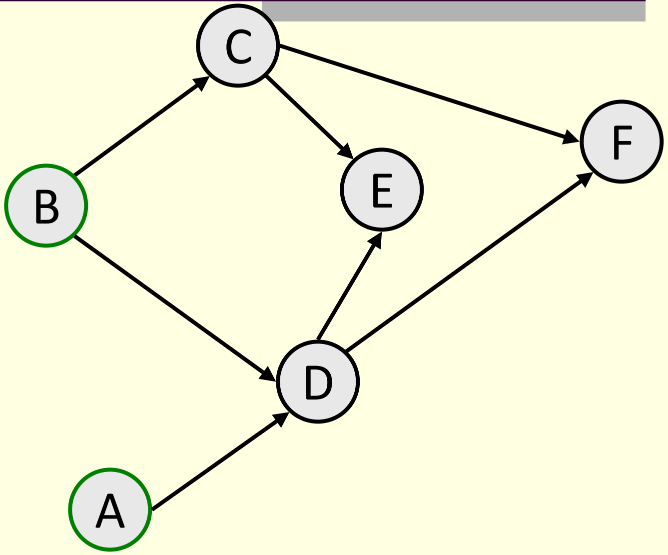


Revised algorithm

- We don't want to literally delete vertices and edges from the graph while trying to topological sort it; so let's revise the algorithm:
  - *map* := {each vertex → its in-degree}.
  - *queue* := {all vertices with in-degree = 0}.
  - *ordering* := { }.
  - Repeat until queue is empty:
    - Dequeue the first vertex *v* from the queue.
    - *ordering* += *v*.
    - Decrease the in-degree of all *v*'s neighbors by 1 in the *map*.
    - *queue* += {any neighbors whose in-degree is now 0}.
  - If all vertices are processed, success.
    Otherwise, there is a cycle.

Topo sort example 2

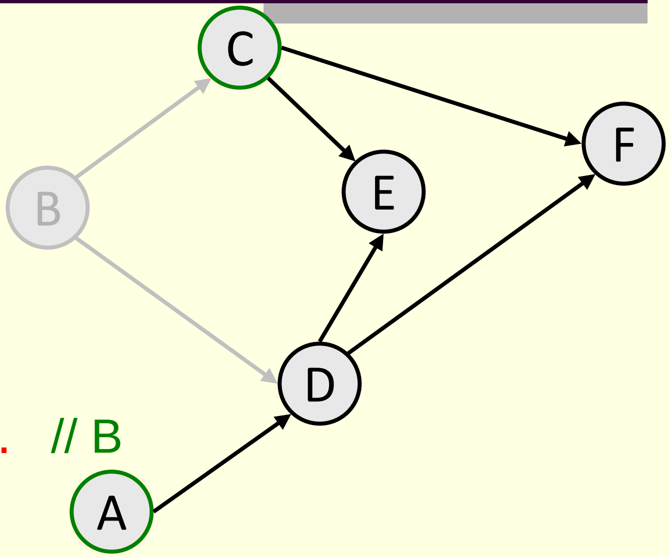
- function topologicalSort():
  - *map* := {each vertex → its in-degree}.
  - *queue* := {all vertices with in-degree = 0}.
  - *ordering* := { }.
  - Repeat until queue is empty:
    - Dequeue the first vertex *v* from the queue.
    - *ordering* += *v*.
    - Decrease the in-degree of all *v*'s neighbors by 1 in the *map*.
    - *queue* += {any neighbors whose in-degree is now 0}.



- map := { A=0, B=0, C=1, D=2, E=2, F=2 }
- queue := { B, A }
- ordering := { }

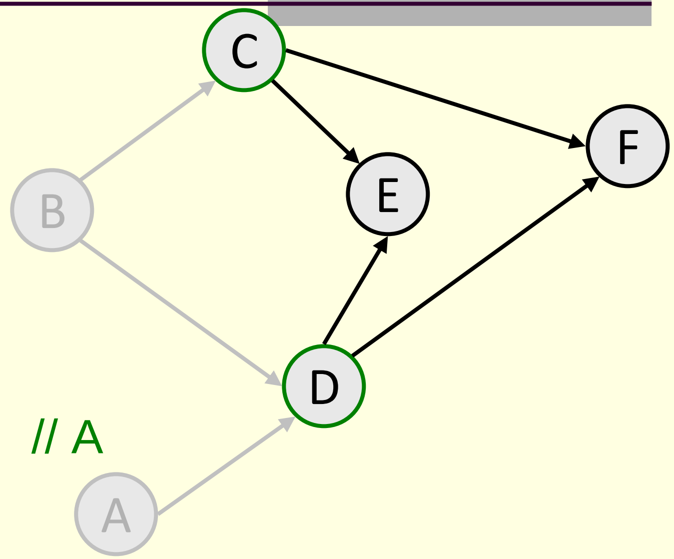
Topo sort example 2

- function topologicalSort():
  - *map* := {each vertex → its in-degree}.
  - *queue* := {all vertices with in-degree = 0}.
  - *ordering* := { }.
  - Repeat until queue is empty:
    - Dequeue the first vertex *v* from the queue. // B
    - *ordering* += *v*.
    - Decrease the in-degree of all *v*'s neighbors by 1 in the *map*. // C, D
    - *queue* += {any neighbors whose in-degree is now 0}.



- map := { A=0, B=0, **C=0**, **D=1**, E=2, F=2 }
- queue := { A, **C** }
- ordering := { **B** }

Topo sort example 2

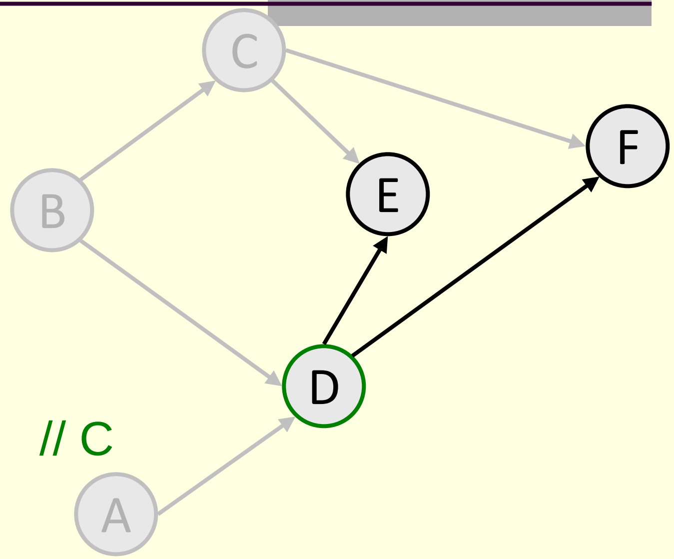


- function topologicalSort():
  - *map* := {each vertex → its in-degree}.
  - *queue* := {all vertices with in-degree = 0}.
  - *ordering* := { }.
  - Repeat until queue is empty:
    - Dequeue the first vertex *v* from the queue. // A
    - *ordering* += *v*.
    - Decrease the in-degree of all *v*'s // D
      neighbors by 1 in the *map*.
    - *queue* += {any neighbors whose in-degree is now 0}.

- map := { A=0, B=0, C=0, **D=0**, E=2, F=2 }
- queue := { C, **D** }
- ordering := { B, **A** }

Topo sort example 2

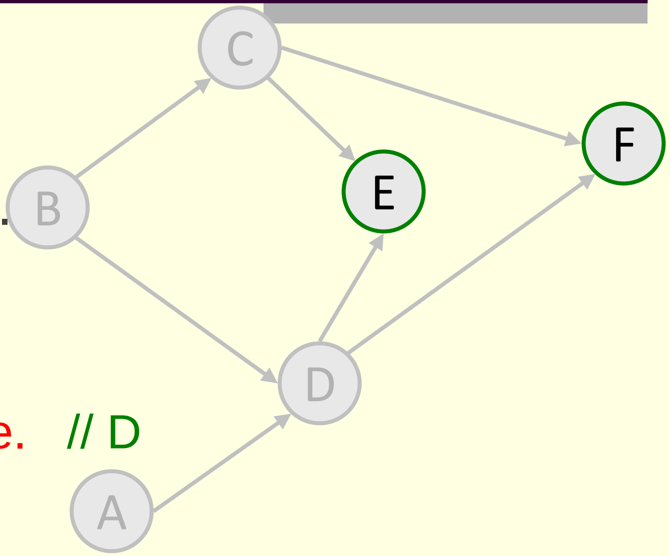
- function topologicalSort():
  - *map* := {each vertex → its in-degree}.
  - *queue* := {all vertices with in-degree = 0}.
  - *ordering* := { }.
  - Repeat until queue is empty:
    - Dequeue the first vertex *v* from the queue. // C
    - *ordering* += *v*.
    - Decrease the in-degree of all *v*'s neighbors by 1 in the *map*. // E, F
    - *queue* += {any neighbors whose in-degree is now 0}.



- map := { A=0, B=0, C=0, D=0, E=**1**, F=**1** }
- queue := { D }
- ordering := { B, A, **C** }

Topo sort example 2

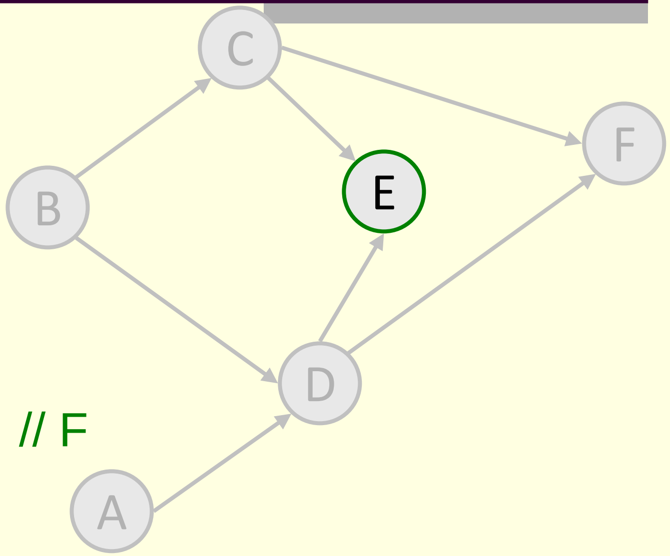
- function topologicalSort():
  - *map* := {each vertex → its in-degree}.
  - *queue* := {all vertices with in-degree = 0}.
  - *ordering* := { }.
  - Repeat until queue is empty:
    - Dequeue the first vertex *v* from the queue. // D
    - *ordering* += *v*.
    - Decrease the in-degree of all *v*'s // F, E
      neighbors by 1 in the *map*.
    - *queue* += {any neighbors whose in-degree is now 0}.



- map := { A=0, B=0, C=0, D=0, E=**0**, F=**0** }
- queue := { **F**, **E** }
- ordering := { B, A, C, **D** }

Topo sort example 2

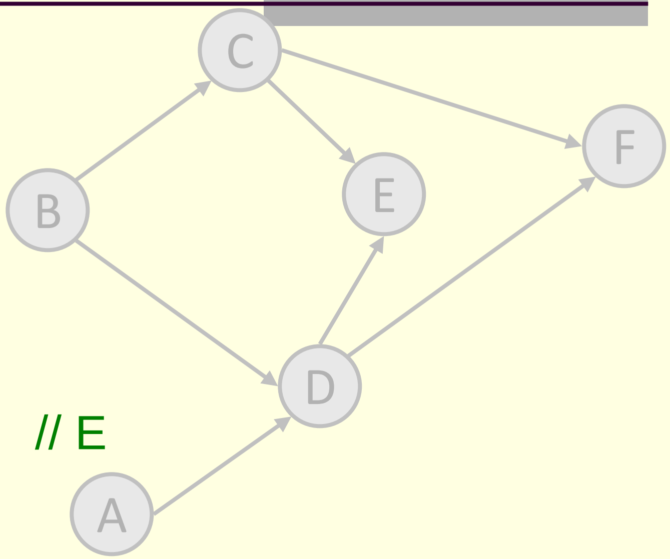
- function topologicalSort():
  - *map* := {each vertex → its in-degree}.
  - *queue* := {all vertices with in-degree = 0}.
  - *ordering* := { }.
  - Repeat until queue is empty:
    - Dequeue the first vertex *v* from the queue. // F
    - *ordering* += *v*.
    - Decrease the in-degree of all *v*'s // none neighbors by 1 in the *map*.
    - *queue* += {any neighbors whose in-degree is now 0}.



- map := { A=0, B=0, C=0, D=0, E=0, F=0 }
- queue := { E }
- ordering := { B, A, C, D, **F** }

Topo sort example 2

- function topologicalSort():
  - *map* := {each vertex → its in-degree}.
  - *queue* := {all vertices with in-degree = 0}.
  - *ordering* := { }.
  - Repeat until queue is empty:
    - Dequeue the first vertex *v* from the queue. // E
    - *ordering* += *v*.
    - Decrease the in-degree of all *v*'s // none neighbors by 1 in the *map*.
    - *queue* += {any neighbors whose in-degree is now 0}.



- map := { A=0, B=0, C=0, D=0, E=0, F=0 }
- queue := { }
- ordering := { B, A, C, D, F, **E** }

Topo sort runtime

- What is the runtime of our topological sort algorithm?
  - (with an "adjacency map" graph internal representation)

  - function topologicalSort():
    - *map* := {each vertex → its in-degree}. // O(*V*)
    - *queue* := {all vertices with in-degree = 0}.
    - *ordering* := { }.
    - Repeat until queue is empty: // O(*V*)
      - Dequeue the first vertex *v* from the queue. // O(*1*)
      - *ordering* += *v*. // O(*1*)
      - Decrease the in-degree of all *v*'s neighbors by 1 in the *map*. // O(*E*) for all passes
      - *queue* += {any neighbors whose in-degree is now 0}.

  - Overall: **O(*V* + *E*)** ; essentially O(*V*) time on a sparse graph (fast!)