CSE 332: Graphs II

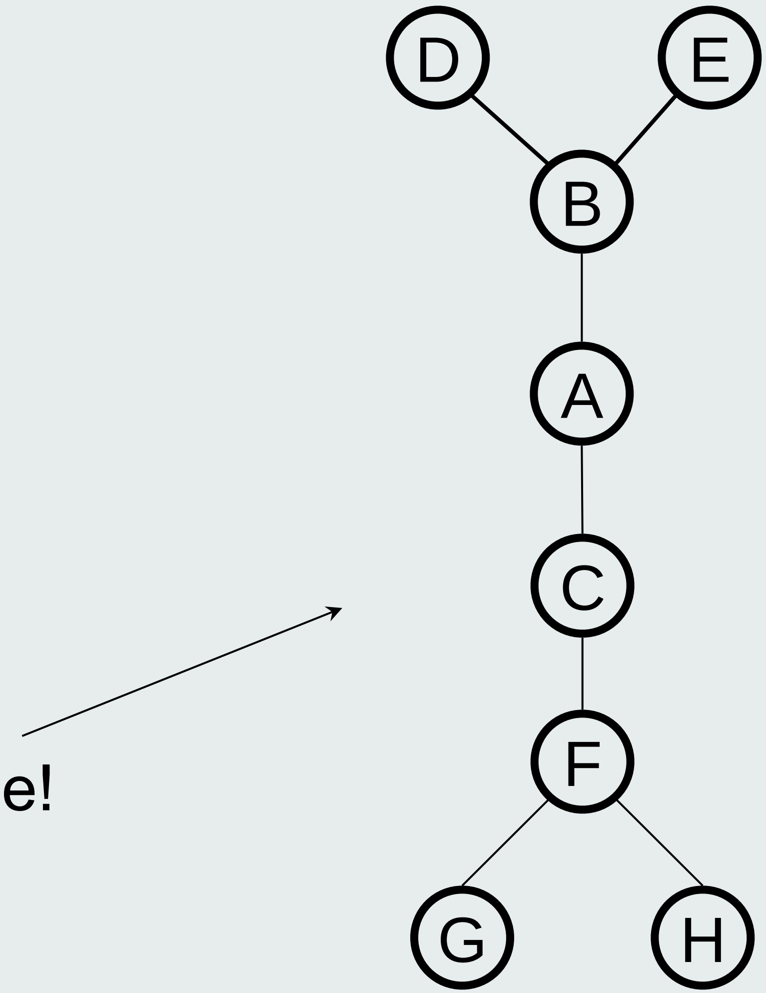
Paul Beame in lieu of Richard Anderson

Spring 2016

Trees as Graphs

A tree is a graph that is:

- *undirected*
- *acyclic*
- *connected*



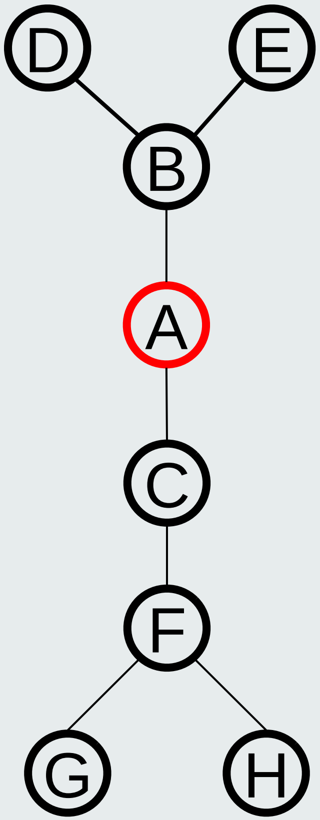
Hey, that doesn't look like a tree!

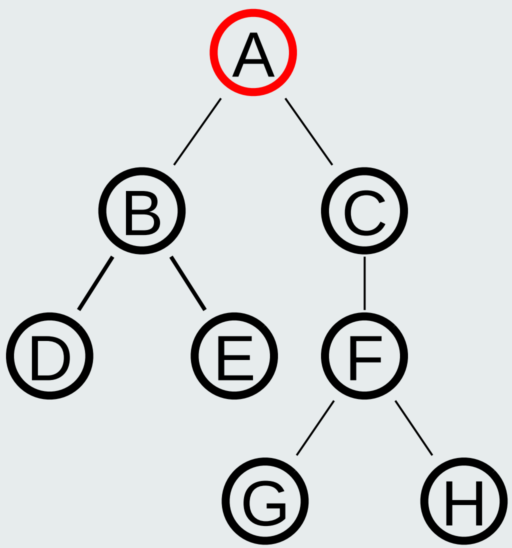
Rooted Trees

We are more accustomed to:

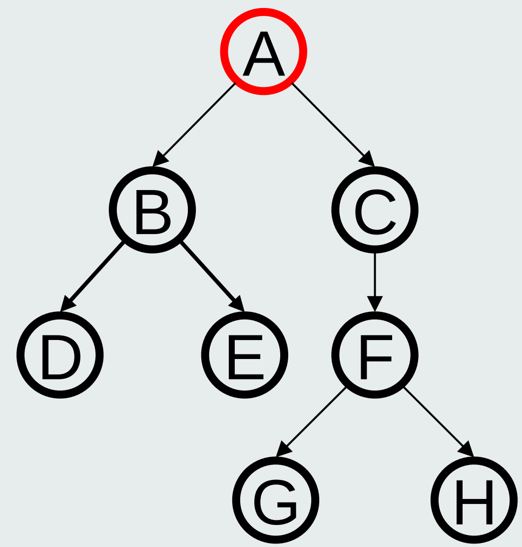
- Rooted trees (a tree node that is “special”)
- Directed edges from parents to children (parent closer to root).

A rooted tree (root indicated in red) drawn two ways





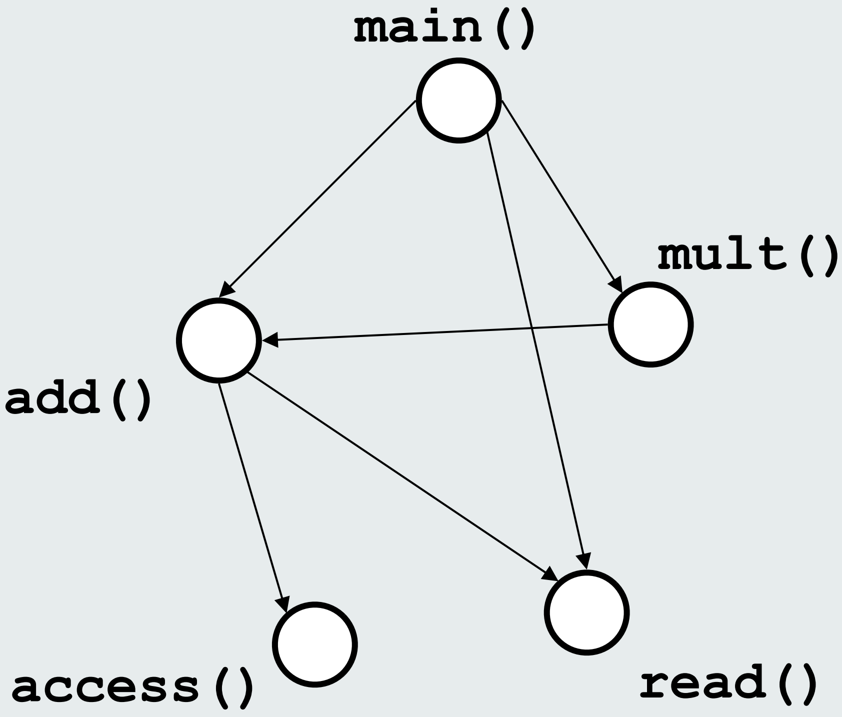
Rooted tree with directed edges from parents to children.



Characteristics of this one?

Directed Acyclic Graphs (DAGs)

DAGs are directed graphs with no (directed) cycles.



Aside: *If program call-graph is a DAG, then all procedure calls can be in-lined*

|E| and |V|

How many edges |E| in a graph with |V| vertices?

What if the graph is directed?

What if it is undirected and connected?

Can the following bounds be simplified?

– Arbitrary graph: $O(|E| + |V|)$
– Arbitrary graph: $O(|E| + |V|^2)$
– Undirected, connected: $O(|E| \log|V| + |V| \log|V|)$

Some (semi-standard) terminology:

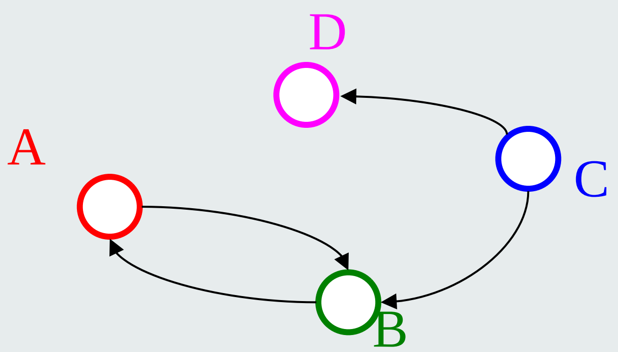
– A graph is *sparse* if it has $O(|V|)$ edges (upper bound).
– A graph is *dense* if it has $\Theta(|V|^2)$ edges.

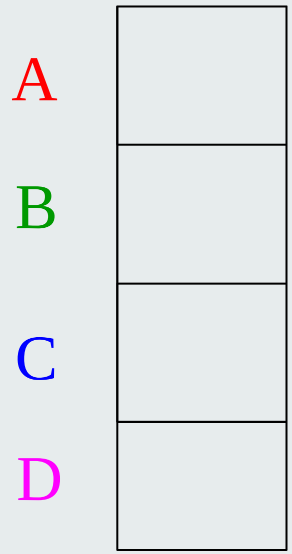
What's the data structure?

- Common query: which edges are adjacent to a vertex

Representation 2: Adjacency List

A list (array) of length `|V|` in which each entry stores a list (linked list) of all adjacent vertices





Runtimes:

Iterate over vertices?

Iterate over edges?

Iterate edges adj. to vertex?

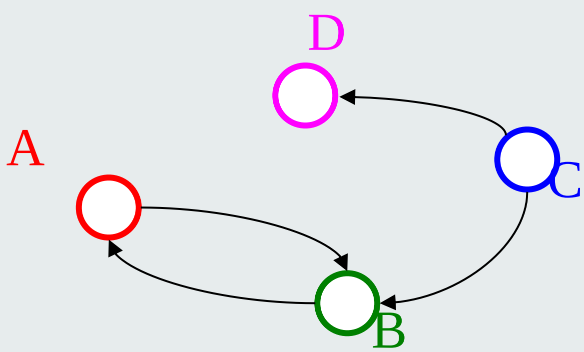
Existence of edge?

Space requirements?

Best for what kinds of graphs?

Representation 1: Adjacency Matrix

A **`|V|×|V|`** matrix **`M`** in which an element **`M[u,v]`** is true if and only if there is an edge from **`u`** to **`v`**



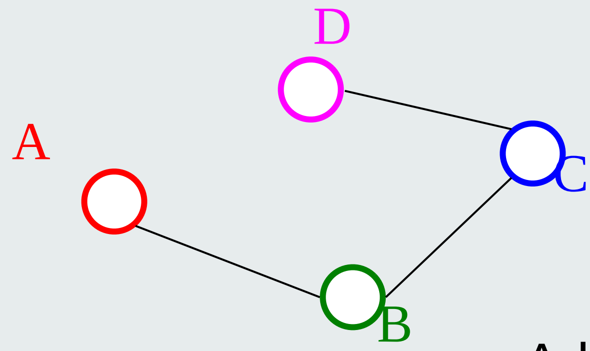
	A	B	C	D
A				
B				
C				
D				

Runtimes:
Iterate over vertices?
Iterate over edges?
Iterate edges adj. to vertex?
Existence of edge?

Space requirements?
Best for what kinds of graphs?

Representing Undirected Graphs

What do these reps look like for an undirected graph?



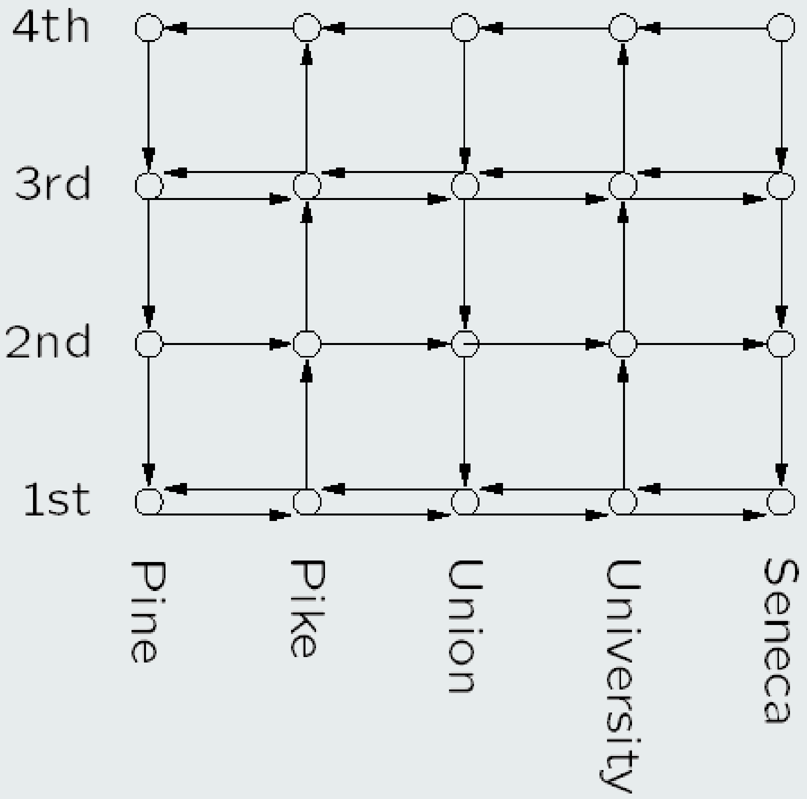
Adjacency matrix:

	A	B	C	D
A				
B				
C				
D				

Adjacency list:

A	
B	
C	
D	

Some Applications:
Bus Routes in Downtown Seattle

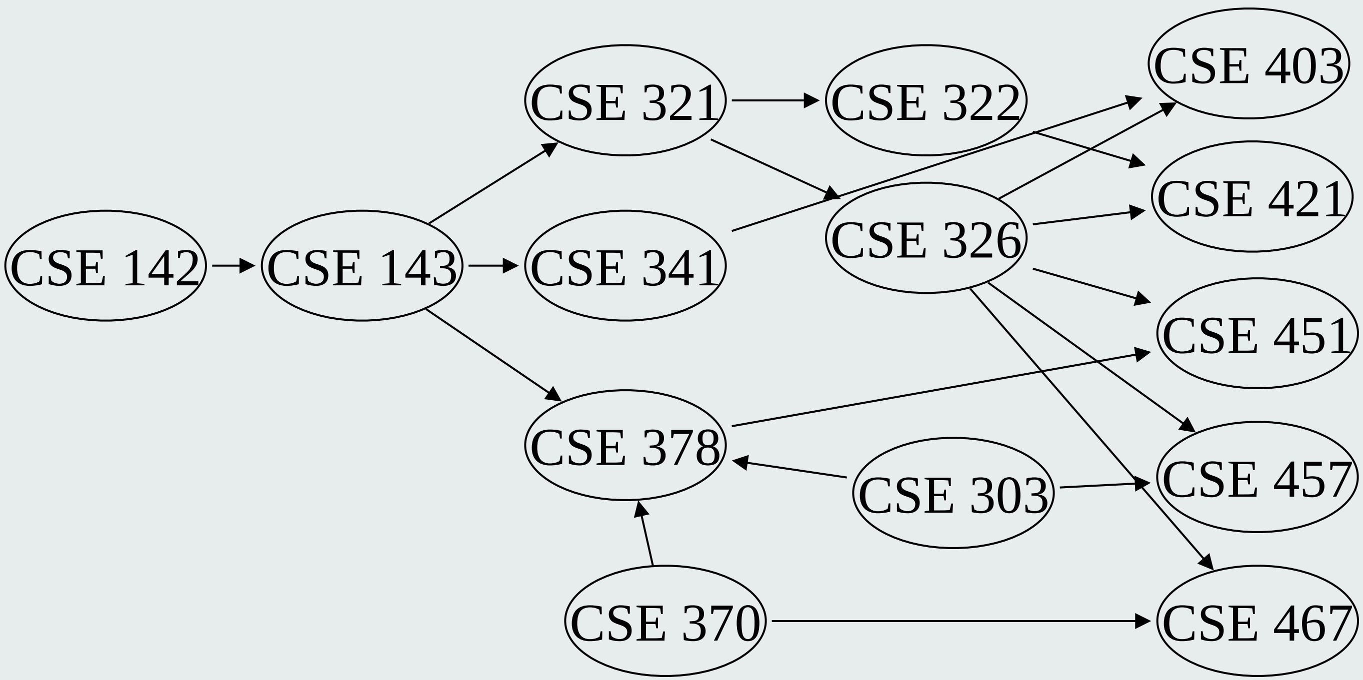


If we're at 3rd and Pine, how can we get to 1st and University using Metro?

How about 4th and Seneca?

Application: Topological Sort

Given a graph, **`G = (V,E)`**, output all the vertices in **`V`** sorted so that no vertex is output before any other vertex with an edge to it.



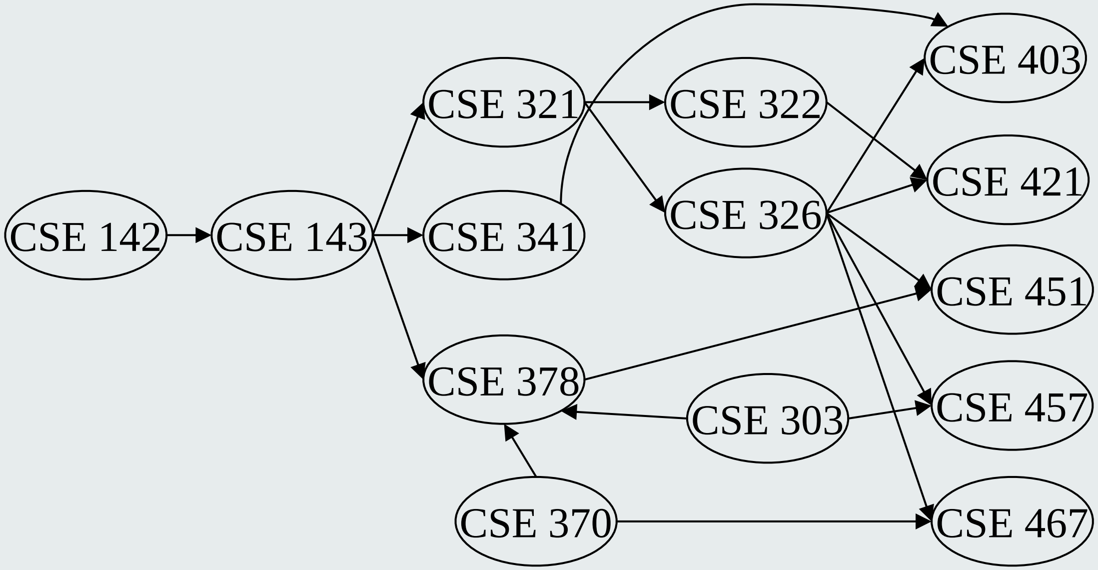
What kind of input graph is allowed?

Is the output unique?

Topological Sort: Take One

1. Label each vertex with its *in-degree* (# inbound edges)
2. **While** there are vertices remaining:
   a. Choose a vertex *v* of *in-degree zero*; output *v*
   b. Reduce the in-degree of all vertices adjacent to *v*
   c. Remove *v* from the list of vertices

Runtime:



142
143
321
341
378
370
322
326

303
403
421
451
457
467

```
void Graph::topsort(){
  Vertex v, w;

labelEachVertexWithItsInDegree();

      for (int counter=0; counter < NUM_VERTICES;
                                     counter++){
            v = findNewVertexOfDegreeZero();

            v.topologicalNum = counter;
            for each w adjacent to v
                  w.indegree--;
  }
}
```

Topological Sort: Take Two

1. Label each vertex with its in-degree
2. Initialize a queue *Q* to contain all in-degree zero vertices
3. While *Q* not empty
   a. *v* = *Q*.dequeue; output *v*
   b. Reduce the in-degree of all vertices adjacent to *v*
   c. If new in-degree of any such vertex *u* is zero *Q*.enqueue(*u*)

Note: could use a stack, list, set, box, … instead of a queue

Runtime:

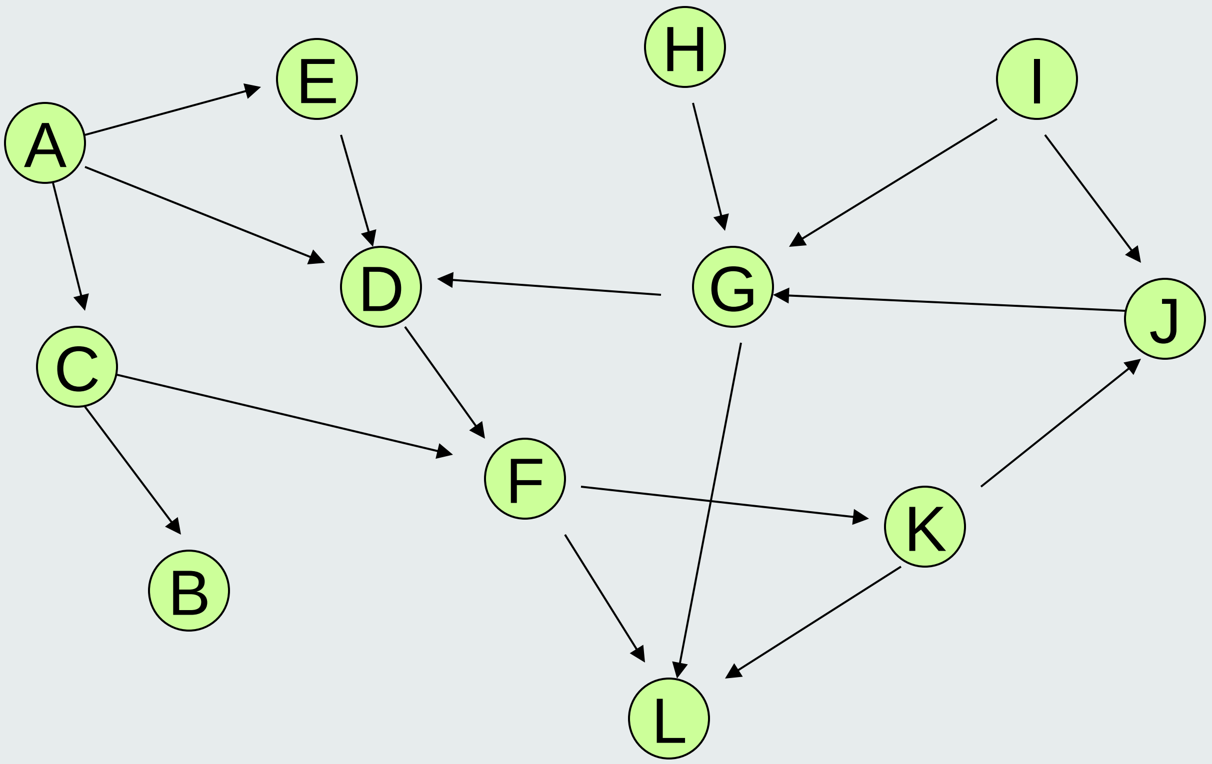
```
void Graph::topsort(){
  Queue q(NUM_VERTICES);
  int counter = 0;
  Vertex v, w;
      labelEachVertexWithItsIn-degree();

  q.makeEmpty();
  for each vertex v
    if (v.indegree == 0)
      q.enqueue(v);

  while (!q.isEmpty()){
    v = q.dequeue();
    v.topologicalNum = ++counter;
    for each w adjacent to v
      if (--w.indegree == 0)
        q.enqueue(w);
  }
}
```

intialize the queue

get a vertex with indegree 0

insert new eligible vertices

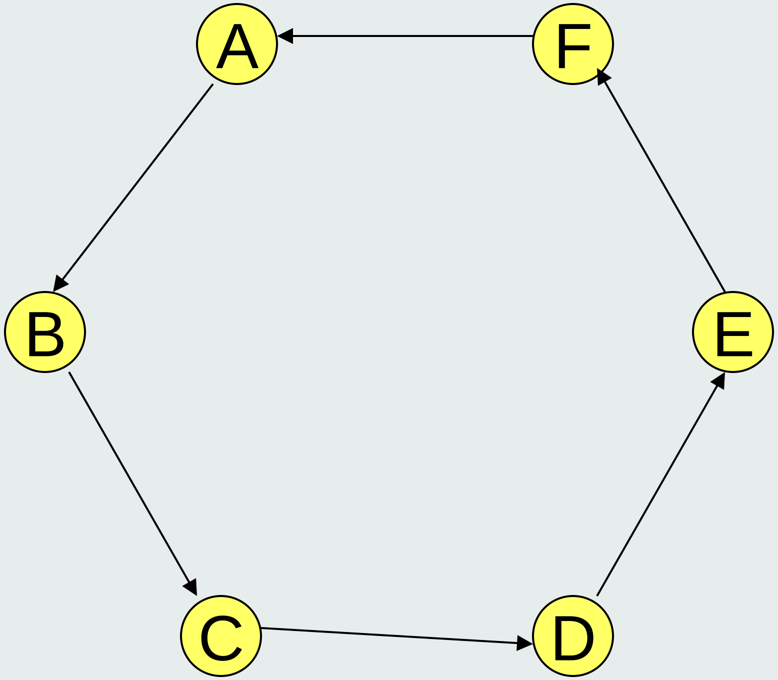
Find a topological order for the following graph



If a graph has a cycle, there is no topological sort

Consider the first vertex on the cycle in the topological sort

It must have an incoming edge



Lemma: If a graph is acyclic, it has a vertex with in degree 0

Proof:

Pick a vertex v_1, if it has in-degree 0 then done

If not, let (v_2, v_1) be an edge, if v_2 has in-degree 0 then done

If not, let (v_3, v_2) be an edge . . .

If this process continues for more than n steps, we have a repeated vertex, so we have a cycle