



CSE 332: Data Structures & Parallelism

Lecture 20: Topological Sort

Ruth Anderson

Winter 2023

Today

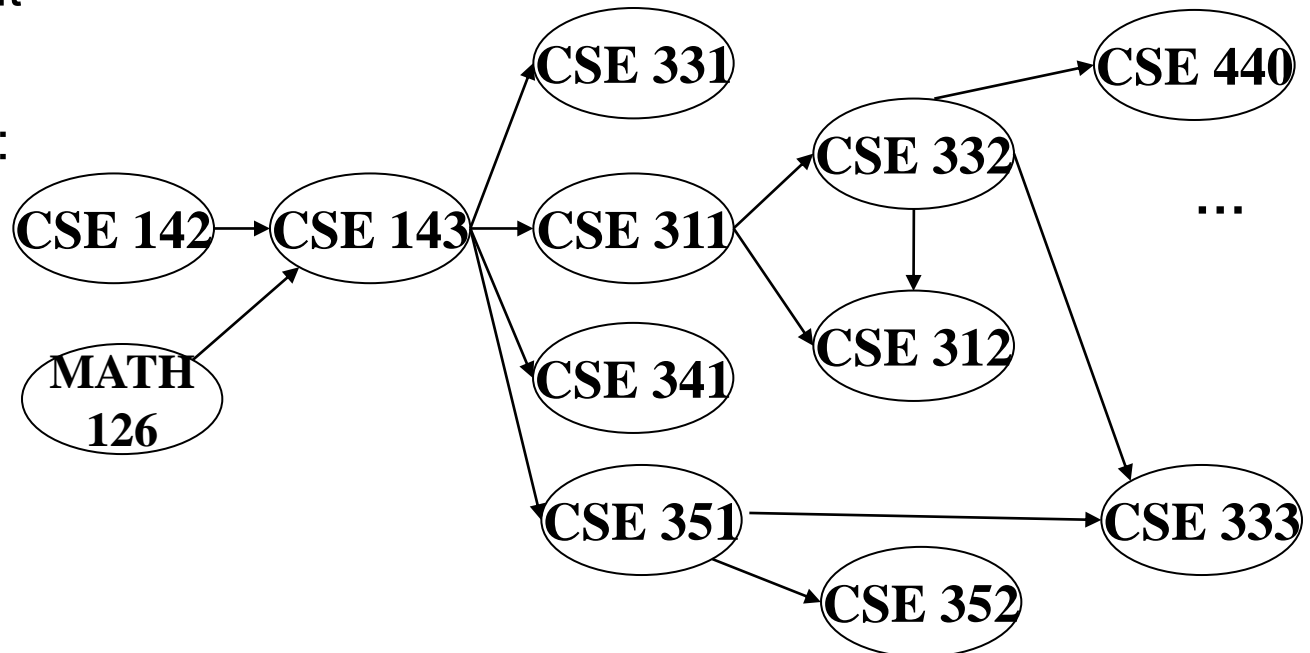
- Graphs
 - Topological Sort

Disclaimer: Do not use for official advising purposes!
(Implies that CSE 332 is a pre-req for CSE 312 – not true)

Topological Sort

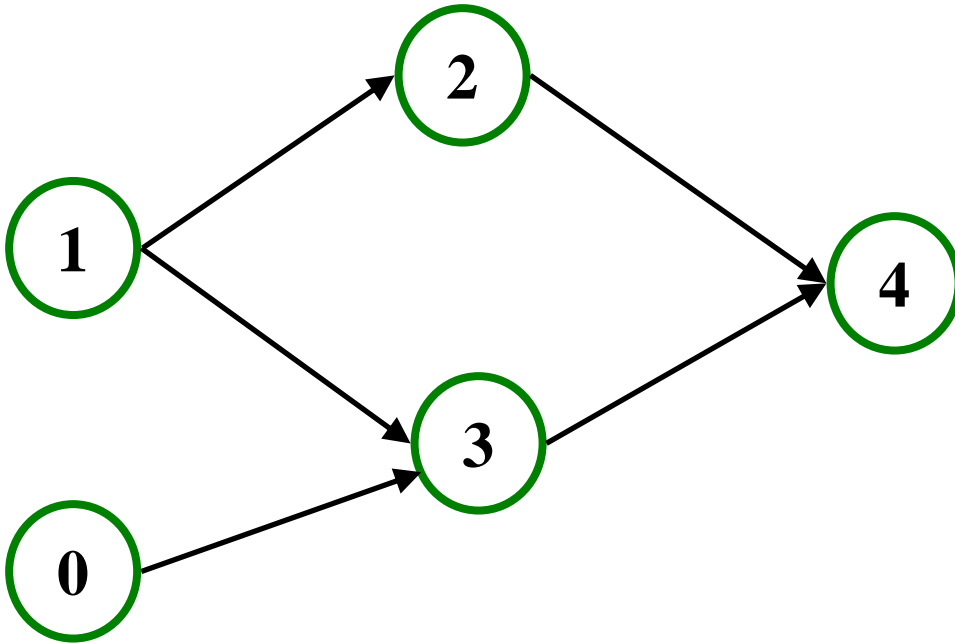
Problem: Given a DAG $G = (V, E)$, output all the vertices in order such that if no vertex appears before any other vertex that has an edge to it

Example input:



Example output:

142, 126, 143, 311, 331, 332, 312, 341, 351, 333, 440, 352



**Valid Topological
Sorts:**

Questions and comments

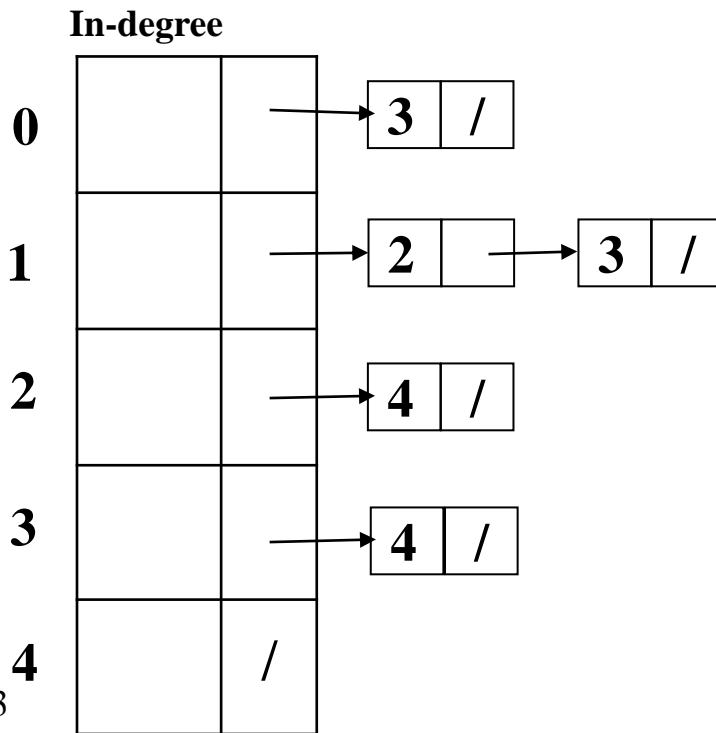
- Why do we perform topological sorts only on DAGs?
- Is there always a unique answer?
- What DAGs have exactly 1 answer?
- Terminology: A DAG represents a **partial order** and a topological sort produces a **total order** that is consistent with it

Topological Sort Uses

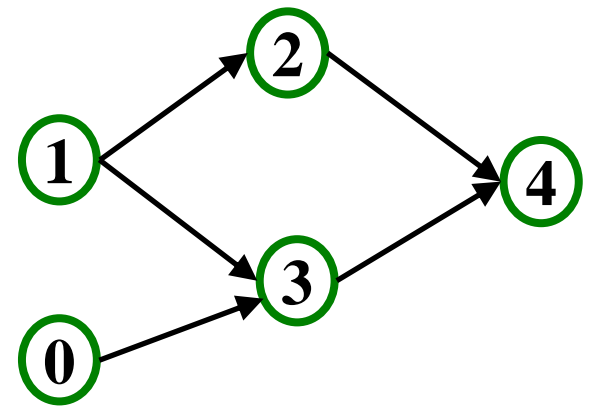
- Figuring out how to finish your degree
- Computing the order in which to recompute cells in a spreadsheet
- Determining the order to compile files using a Makefile
- In general, taking a dependency graph and coming up with an order of execution

A First Algorithm for Topological Sort

1. Label (“mark”) each vertex with its in-degree
 - Think “write in a field in the vertex”
 - Could also do this via a data structure (e.g., array) on the side
2. While there are vertices not yet output:
 - a) Choose a vertex v labeled with in-degree of 0
 - b) Output v and *conceptually* remove it from the graph
 - c) For each vertex w adjacent to v (i.e. w such that (v,w) in \mathbf{E}), decrement the in-degree of w

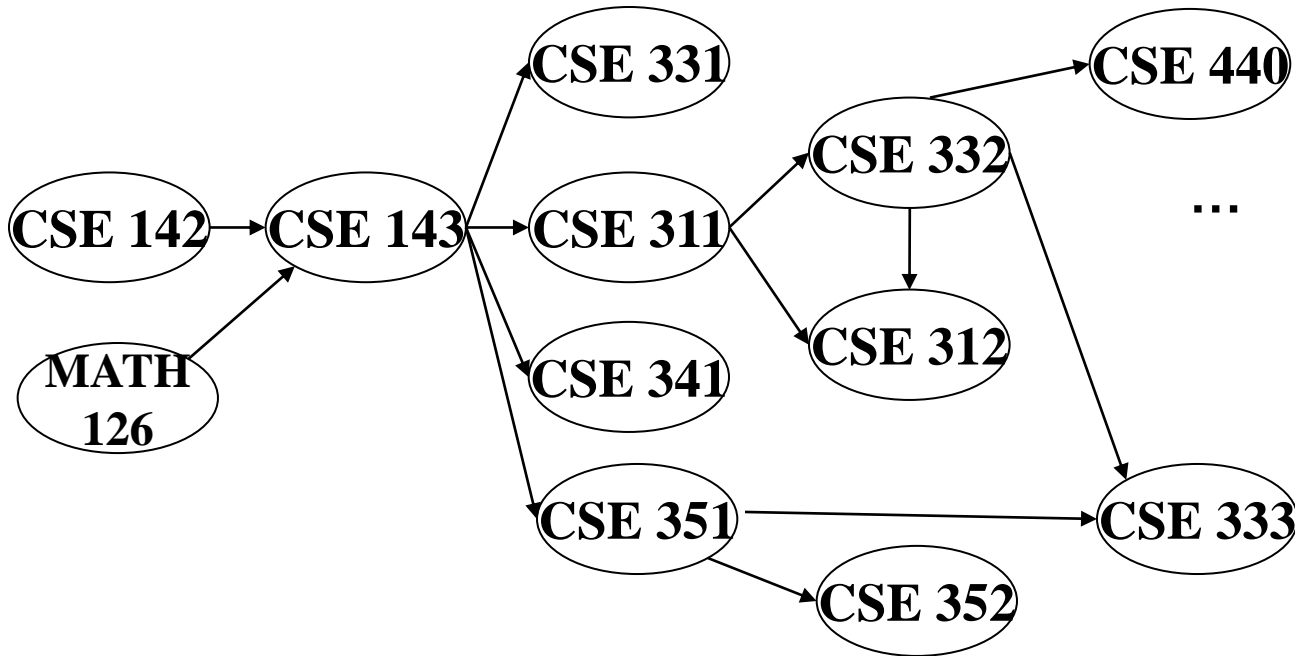


decrement the in-degree of w



Example

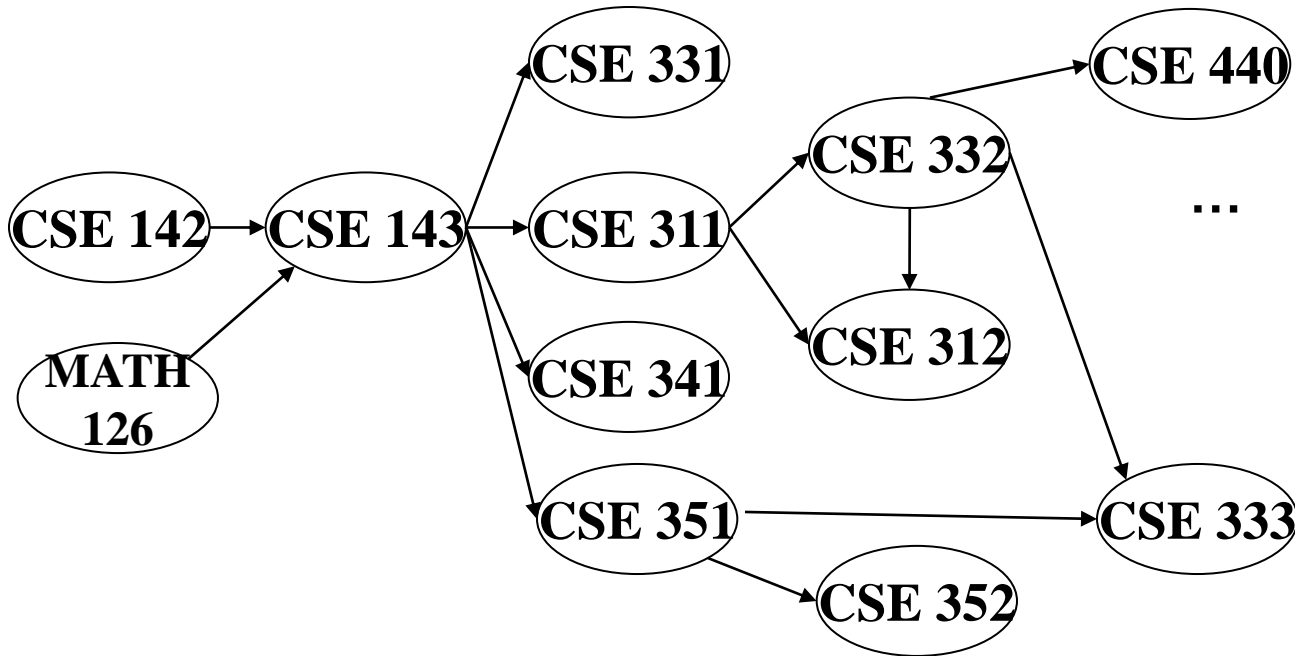
Output:



Node:	126	142	143	311	312	331	332	333	341	351	352	440
Removed?												
In-degree:	0	0	2	1	2	1	1	2	1	1	1	1

Example

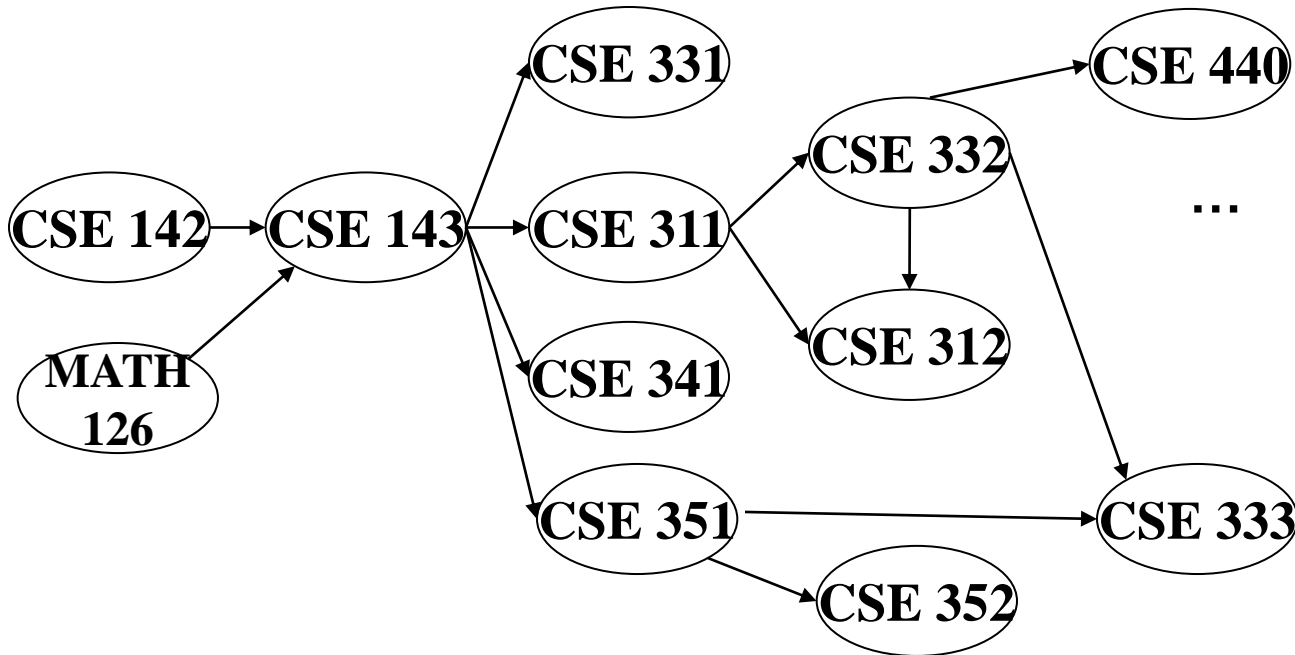
Output: 126



Node:	126	142	143	311	312	331	332	333	341	351	352	440
Removed?	x											
In-degree:	0	0	2	1	2	1	1	2	1	1	1	1
			1									

Example

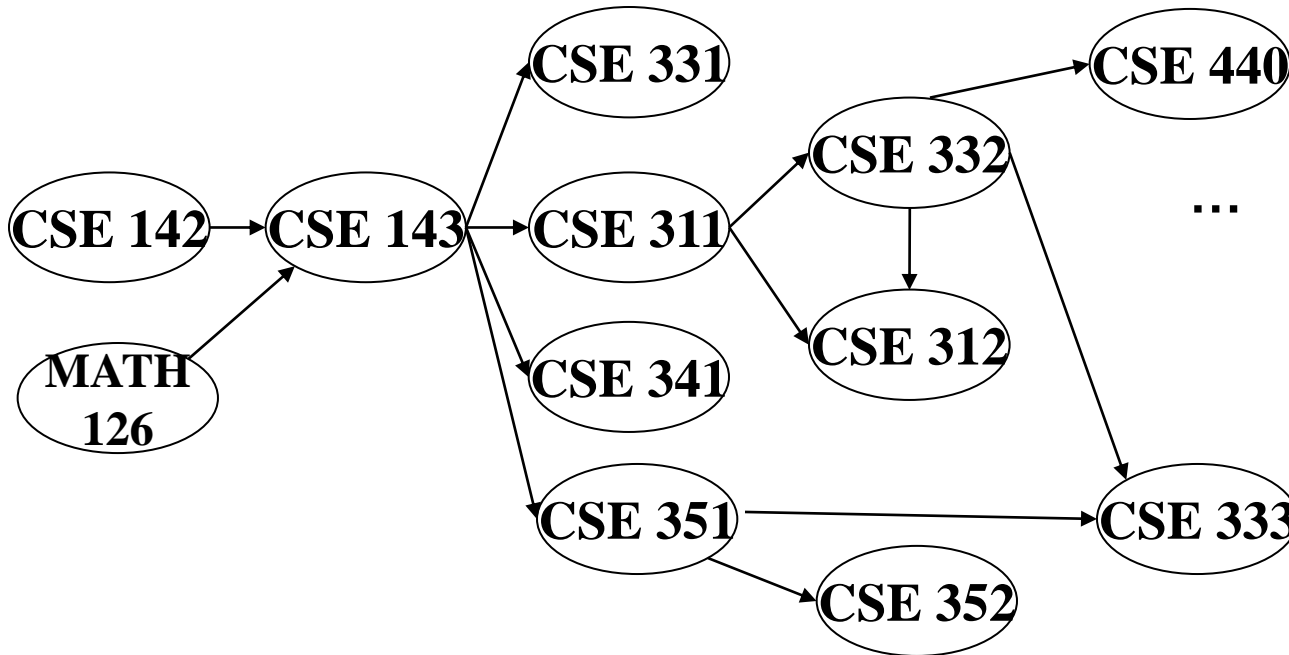
Output: 126
142



Node:	126	142	143	311	312	331	332	333	341	351	352	440
Removed?	x	x										
In-degree:	0	0	2	1	2	1	1	2	1	1	1	1
			1									
			0									

Example

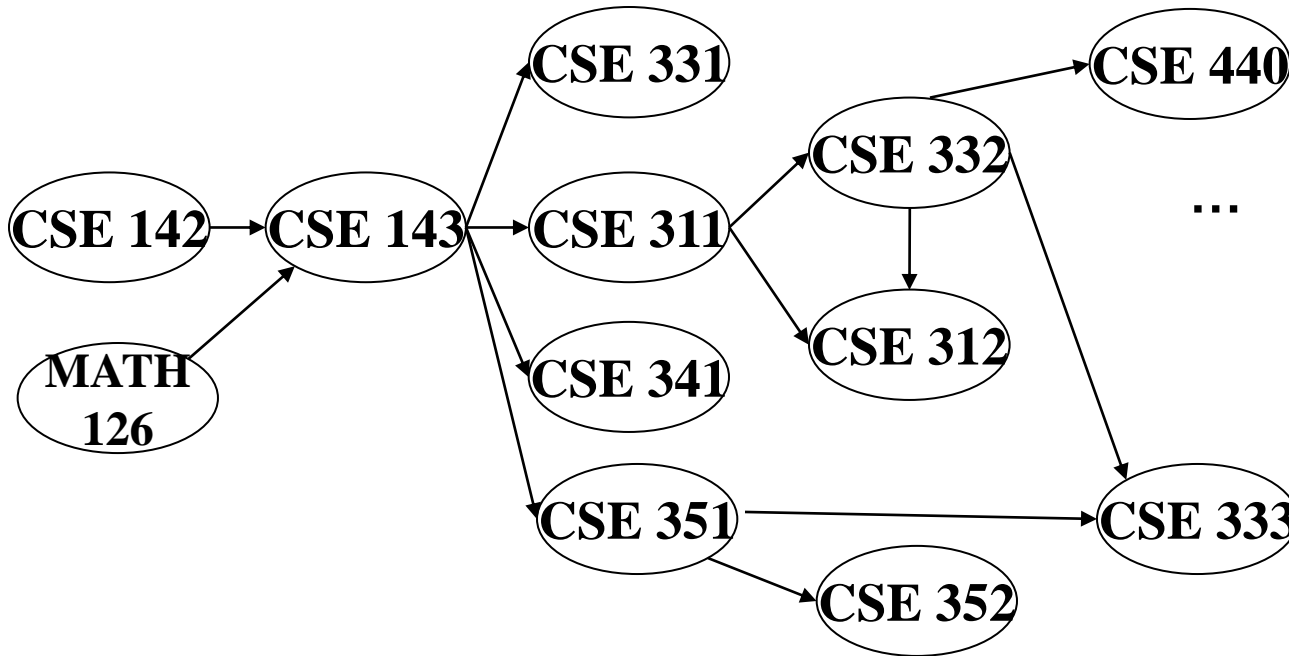
Output: 126
142
143



Node:	126	142	143	311	312	331	332	333	341	351	352	440
Removed?	x	x	x									
In-degree:	0	0	2	1	2	1	1	2	1	1	1	1
			1	0		0			0	0		
			0									

Example

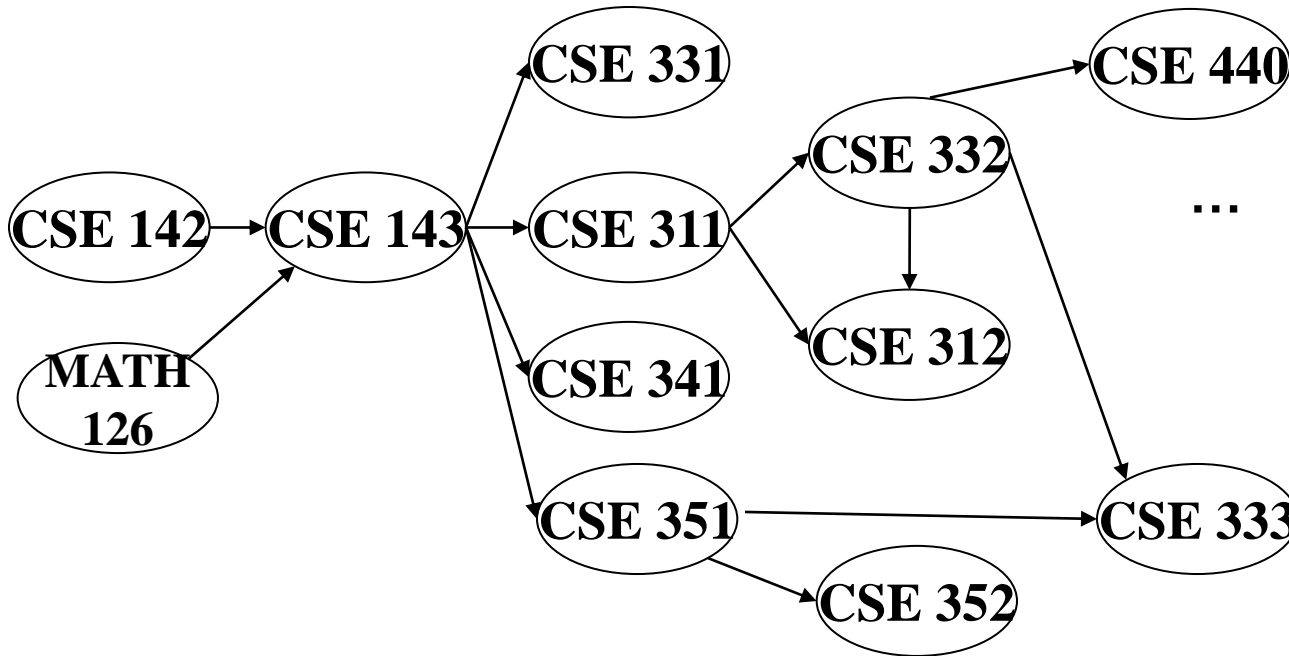
Output: 126
142
143
311
...



Node:	126	142	143	311	312	331	332	333	341	351	352	440
Removed?	x	x	x	x								
In-degree:	0	0	2	1	2	1	1	2	1	1	1	1
			1	0	1	0	0		0	0		
			0									

Example

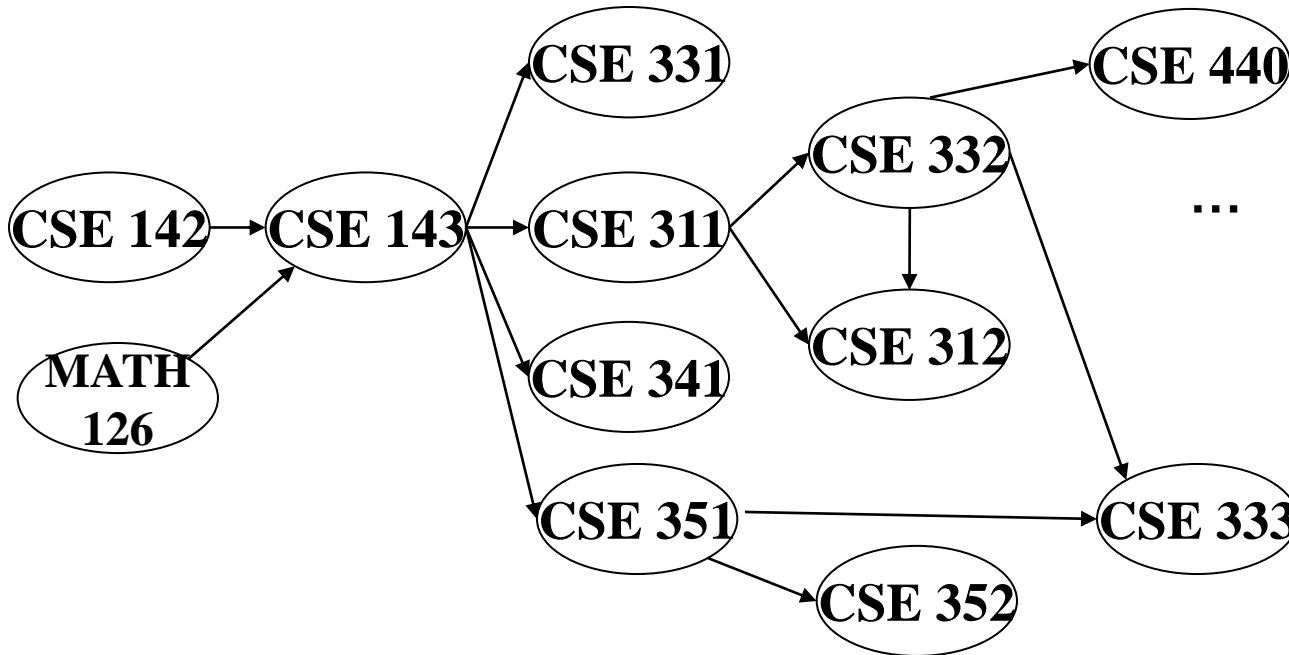
Output: 126
 142
 143
 311
 331



Node:	126	142	143	311	312	331	332	333	341	351	352	440
Removed?	x	x	x	x		x						
In-degree:	0	0	2	1	2	1	1	2	1	1	1	1
			1	0	1	0	0		0	0		
			0									

Example

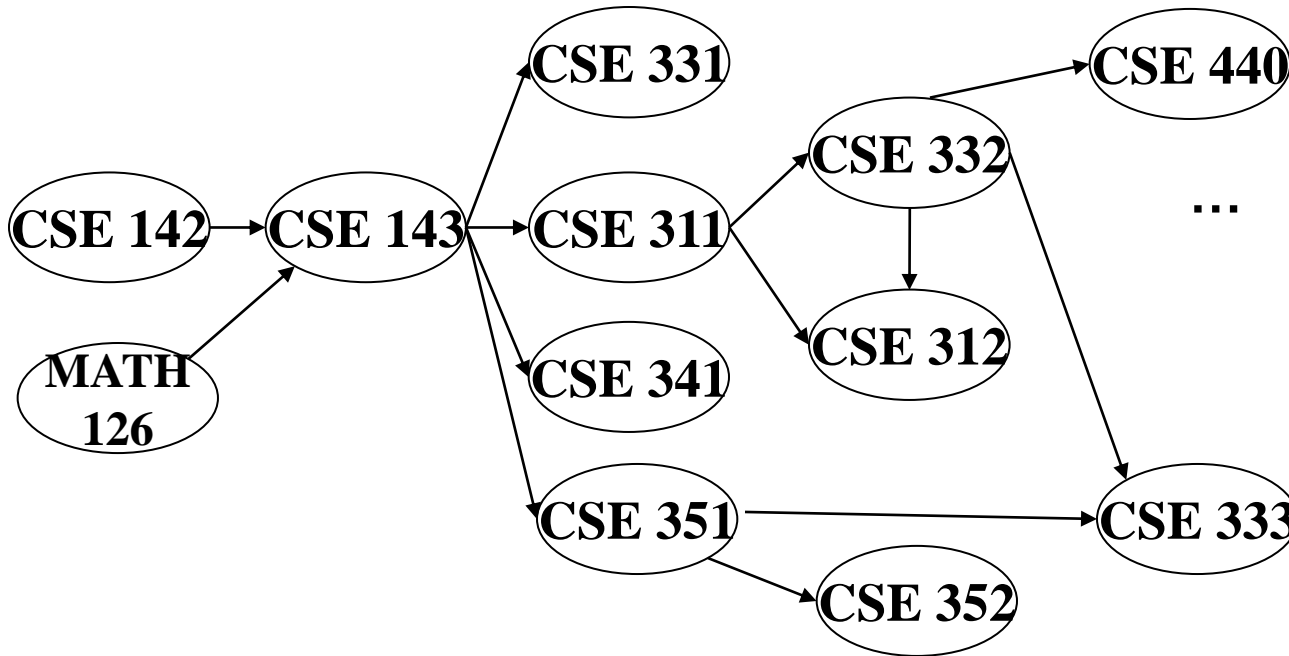
Output: 126
 142
 143
 311
 331
 332



Node:	126	142	143	311	312	331	332	333	341	351	352	440
Removed?	x	x	x	x		x	x					
In-degree:	0	0	2	1	2	1	1	2	1	1	1	1
			1	0	1	0	0	1	0	0		0
			0		0							

Example

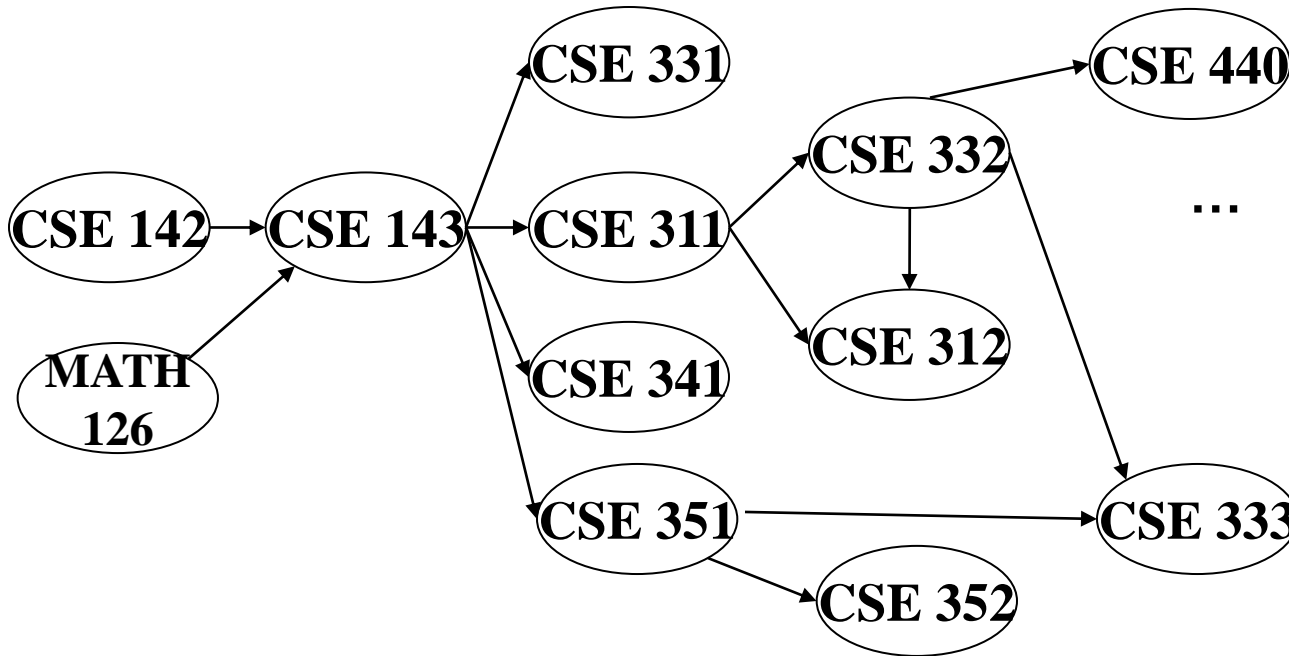
Output: 126
 142
 143
 311
 331
 332
 312



Node:	126	142	143	311	312	331	332	333	341	351	352	440
Removed?	x	x	x	x	x	x	x					
In-degree:	0	0	2	1	2	1	1	2	1	1	1	1
			1	0	1	0	0	1	0	0		0
			0		0							

Example

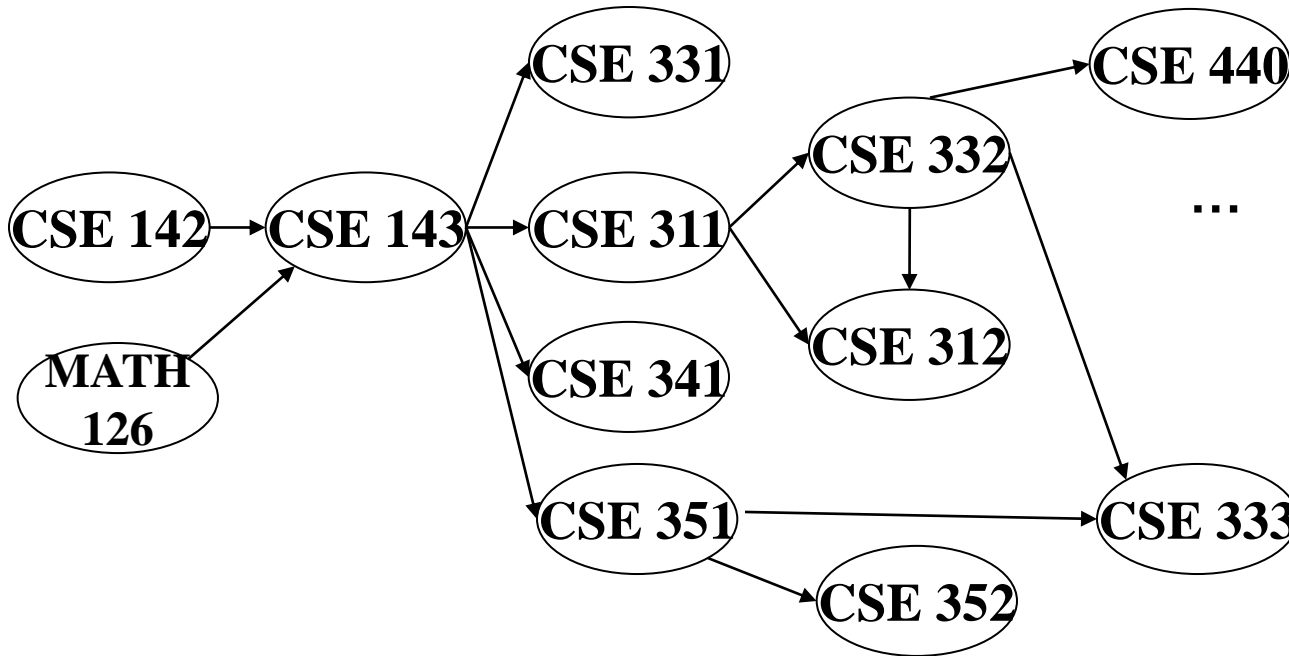
Output: 126
 142
 143
 311
 331
 332
 312
 341



Node:	126	142	143	311	312	331	332	333	341	351	352	440
Removed?	x	x	x	x	x	x	x		x			
In-degree:	0	0	2	1	2	1	1	2	1	1	1	1
			1	0	1	0	0	1	0	0		0
			0		0							

Example

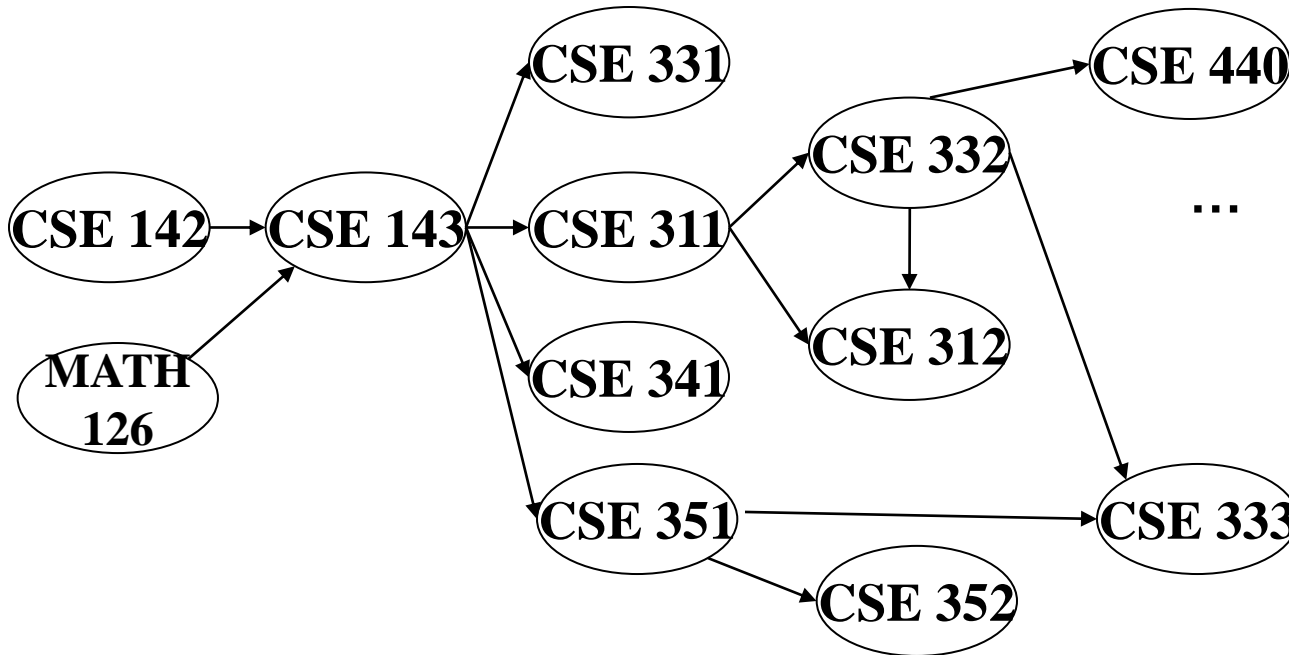
Output: 126
 142
 143
 311
 331
 332
 312
 341
 351



Node:	126	142	143	311	312	331	332	333	341	351	352	440
Removed?	x	x	x	x	x	x	x		x	x		
In-degree:	0	0	2	1	2	1	1	2	1	1	1	1
			1	0	1	0	0	1	0	0	0	0
			0		0			0				

Example

Output: 126
 142
 143
 311
 331
 332
 312
 341
 351
 333
 352
 440



Node:	126	142	143	311	312	331	332	333	341	351	352	440
Removed?	x	x	x	x	x	x	x	x	x	x	x	x
In-degree:	0	0	2	1	2	1	1	2	1	1	1	1
			1	0	1	0	0	1	0	0	0	0
			0		0			0				

A couple of things to note

- Needed a vertex with in-degree of 0 to start
 - No cycles
- Ties between vertices with in-degrees of 0 can be broken arbitrarily
 - Potentially many different correct orders

Topological Sort: Running time?

```
labelEachVertexWithItsInDegree();  
for(ctr=0; ctr < numVertices; ctr++){  
    v = findNewVertexOfDegreeZero();  
    put v next in output  
    for each w adjacent to v  
        w.indegree--;  
}
```

Doing better

The trick is to avoid searching for a zero-degree node every time!

- Keep the “pending” zero-degree nodes in a list, stack, queue, box, table, or something
- Order we process them affects output but not correctness or efficiency provided add/remove are both $O(1)$

Using a queue:

1. Label each vertex with its in-degree, enqueue 0-degree nodes
2. While queue is not empty
 - a) $\mathbf{v} = \text{dequeue}()$
 - b) Output \mathbf{v} and remove it from the graph
 - c) For each vertex \mathbf{w} adjacent to \mathbf{v} (i.e. \mathbf{w} such that (\mathbf{v}, \mathbf{w}) in \mathbf{E}), decrement the in-degree of \mathbf{w} , if new degree is 0, enqueue it

Topological Sort(optimized): Running time?

```
labelAllAndEnqueueZeros();  
for(ctr=0; ctr < numVertices; ctr++){  
    v = dequeue();  
    put v next in output  
    for each w adjacent to v {  
        w.indegree--;  
        if(w.indegree==0)  
            enqueue(w);  
    }  
}
```