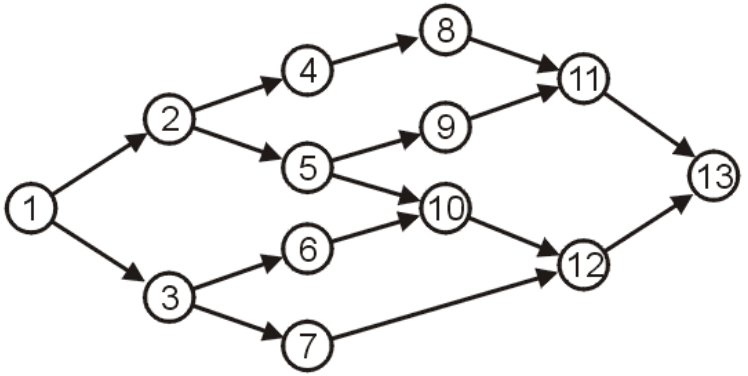
Topological sort



Carlos Moreno
cmoreno@uwaterloo.ca
EIT-4103

https://ece.uwaterloo.ca/~cmoreno/ece250

These slides, the course material, and course web site are based on work by Douglas W. Harder

Topological sort

Standard reminder to set phones to silent/vibrate mode, please!

Topological sort

- During today's class, we will:
  - Look at Topological sort, a common and useful operation with Directed Acyclic Graphs (DAGs)
  - Discuss an algorithm to implement this operation.
  - Briefly talk about some of its applications.

Topological sort

- The basic idea of a topological sort is the following:
  - Given a DAG, in which we could see adjacencies as representing a pre-requisite task, we want to place the vertices in sequence.
  - The sequence must be one in which all dependencies on pre-requisites are satisfied.
  - Such a sequential arrangement of the vertices is called a *topological sort* of the DAG.

Topological sort

- A simple and obvious application is:
  - We have a DAG where vertices are tasks that we want to perform (e.g., part of a compiler's code generation subsystem).
  - We need to execute all the tasks, but we can only do one at a time.
  - A topological sort gives a valid sequence of executed tasks — no task can proceed until all pre-requisite tasks have completed.

Topological sort

- Formally, we would define a topological sort as follows:
  - Let $G = (V, E)$ be a DAG, with $V = \{v_1, v_2, \cdots, v_{n-1}, v_n\}$

Topological sort

- Formally, we would define a topological sort as follows:
  - Let $G = (V, E)$ be a DAG, with $V = \{v_1, v_2, \cdots, v_{n-1}, v_n\}$
  - Then, a topological sort is a sequence containing all the elements in V, $\{v_{k_1}, v_{k_2}, \cdots, v_{k_{n-1}} v_{k_n}\}$ such that for all i, j ($0 \leqslant i, j \leqslant n$), if there exists a path from v_{k_i} to v_{k_j}, then $i < j$.

Topological sort

- Formally, we would define a topological sort as follows:
  - Let $G = (V, E)$ be a DAG, with $V = \{v_1, v_2, \cdots, v_{n-1}, v_n\}$
  - Then, a topological sort is a sequence containing all the elements in V, $\{v_{k_1}, v_{k_2}, \cdots, v_{k_{n-1}} v_{k_n}\}$ such that for all i, j ($0 \leqslant i, j \leqslant n$), if there exists a path from v_{k_i} to v_{k_j}, then $i < j$.
    - Simply put, if there is a path from vertex v to vertex w, then v appears before w in the output sequence.

Topological sort

- Question: why is the notion specific to DAGs?

Topological sort

- In fact, let's look at (and prove) this interesting fact:
  - A directed graph is a DAG if and only if it has a topological sort.

Topological sort

- In fact, let's look at (and prove) this interesting fact:
  - A directed graph is a DAG if and only if it has a topological sort.
  - Proof:

    We observe that there are two independent statements to prove:
    - A DAG has a topological sort
    - If a directed graph has a topological sort, then it is a DAG

Topological sort

- In fact, let's look at (and prove) this interesting fact:
  - A directed graph is a DAG if and only if it has a topological sort.
  - Proof:

    We observe that there are two independent statements to prove:
    - A DAG has a topological sort
    - If a directed graph has a topological sort, then it is a DAG
  - (this is a normal aspect of if-and-only-if statements; proving them is really proving two statements)

Topological sort

- Let's start with the easy one:
  - If a graph has a topological sort, then it is a DAG.
  - Proof: (by contrapositive — that is, we prove the statement "if a graph is not a DAG, it can not have a topological sort")

    Can you complete this part of the proof? (we'll look at it in class, of course!)

Topological sort

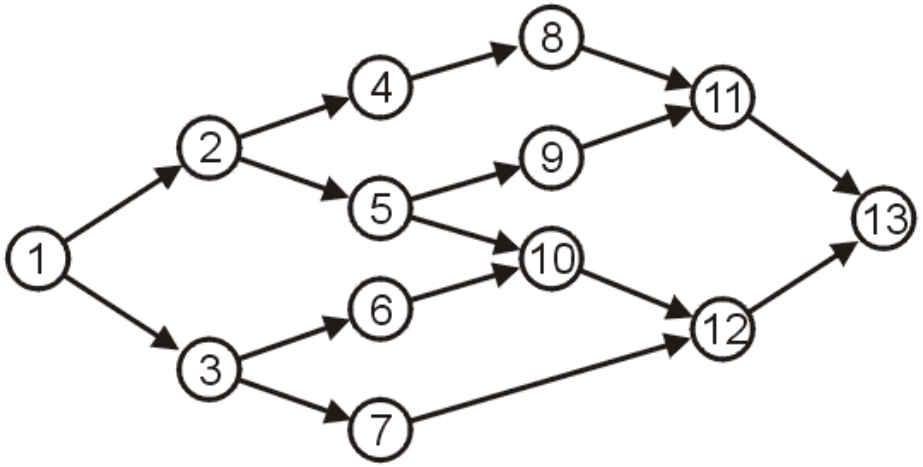
- A somewhat more “obvious” observation:
  - A topological sort is not necessarily unique. (yes?)

Topological sort

- Next, let's look at an algorithm to obtain a topological sort given a DAG.
  - The idea is that any vertex with in-degree 0 can be the first one in a topological sort.
  - We can look at it as follows: each time that we output one of those in-degree 0 vertices, we remove it from the graph.
    - That would in turn lead to creating additional vertices with in-degree 0, which we can now output.

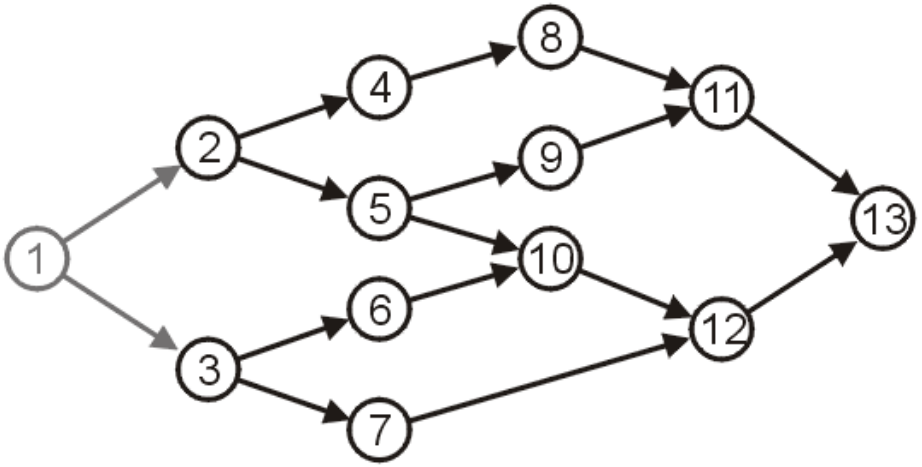
Topological sort

- An example:
  - There's only one vertex with in-degree 0 (vertex 1), so we start with that one (and think of it as removed from the graph):



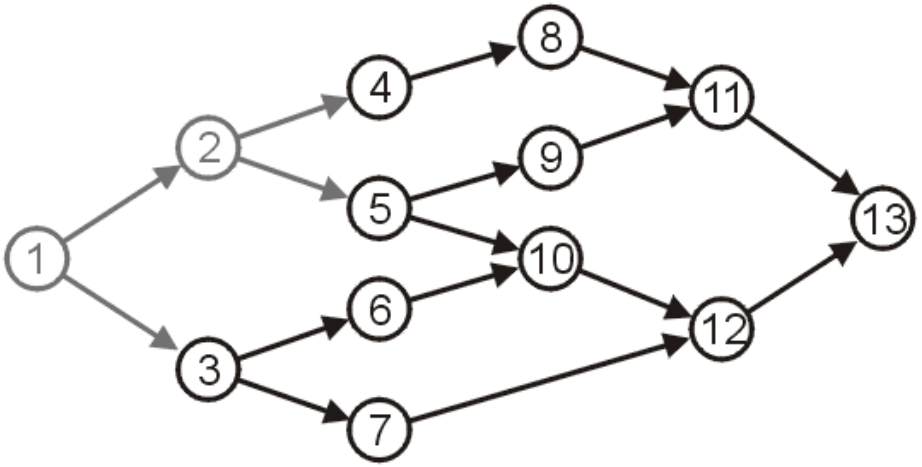
Topological sort

- An example:
  - As we “remove” 1, now 2 and 3 have in-degree 0, so the topological sort could continue with either one of these — we'll choose 2:



Topological sort

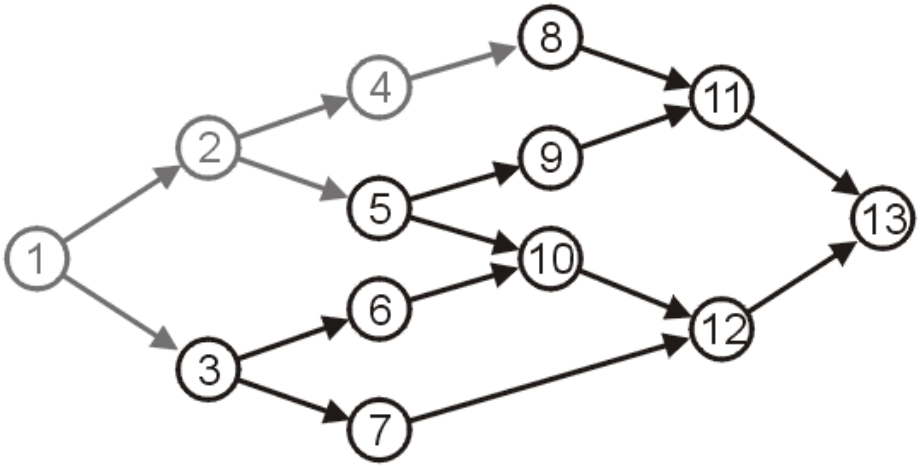
- An example:
  - As we “remove” 1, now 2 and 3 have in-degree 0, so the topological sort could continue with either one of these — we'll choose 2:



1, 2

Topological sort

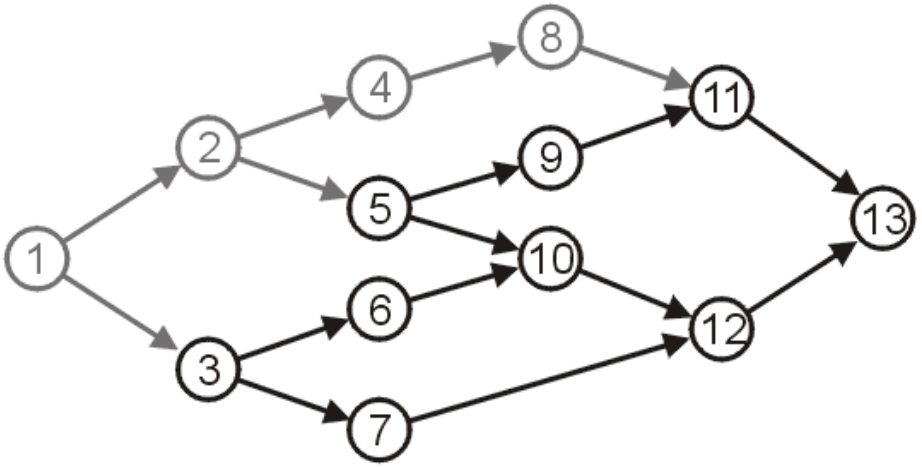
- An example:
  - As we “remove” 1, now 2 and 3 have in-degree 0, so the topological sort could continue with either one of these — we'll choose 2:



1, 2, 4

Topological sort

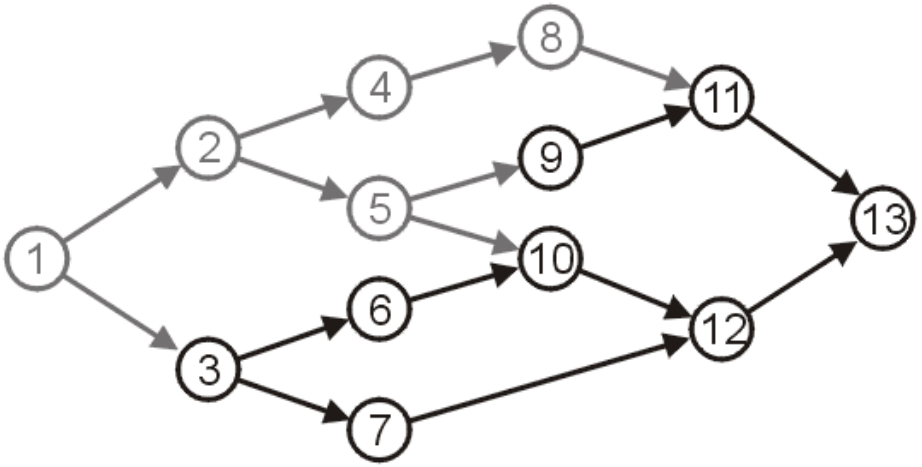
- An example:
  - As we “remove” 1, now 2 and 3 have in-degree 0, so the topological sort could continue with either one of these — we'll choose 2:



1, 2, 4, 8

Topological sort

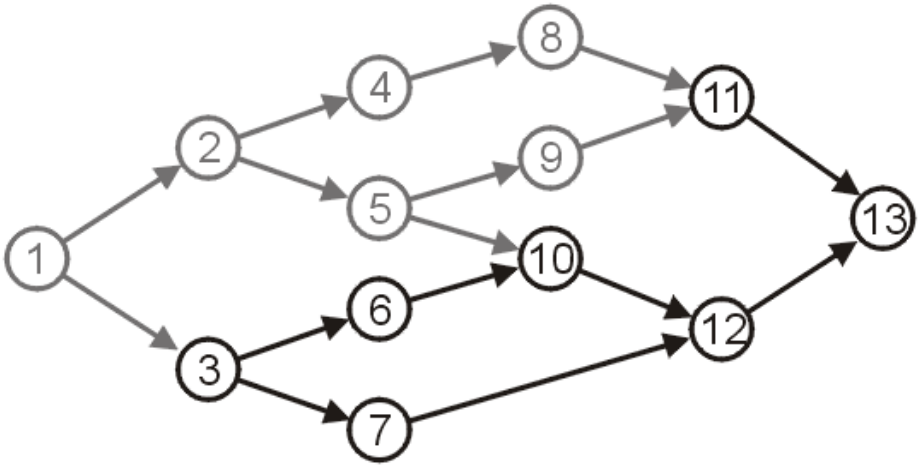
- An example:
  - As we “remove” 1, now 2 and 3 have in-degree 0, so the topological sort could continue with either one of these — we'll choose 2:



1, 2, 4, 8, 5

Topological sort

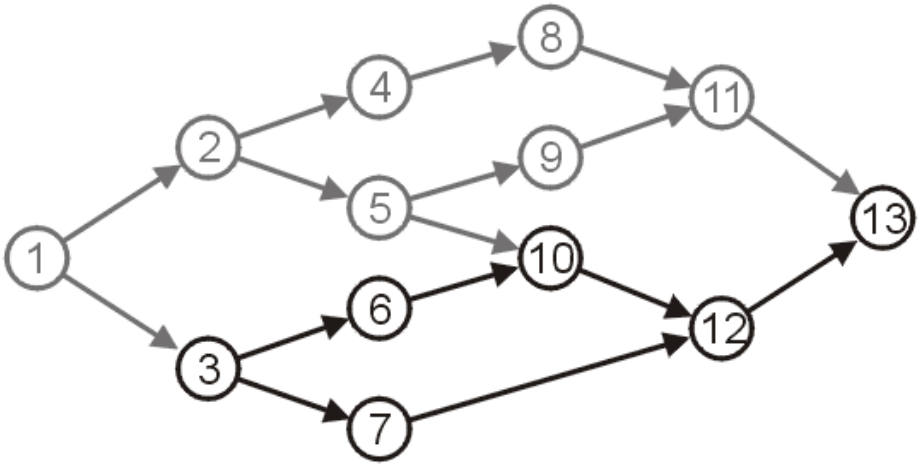
- An example:
  - As we “remove” 1, now 2 and 3 have in-degree 0, so the topological sort could continue with either one of these — we'll choose 2:



1, 2, 4, 8, 5, 9

Topological sort

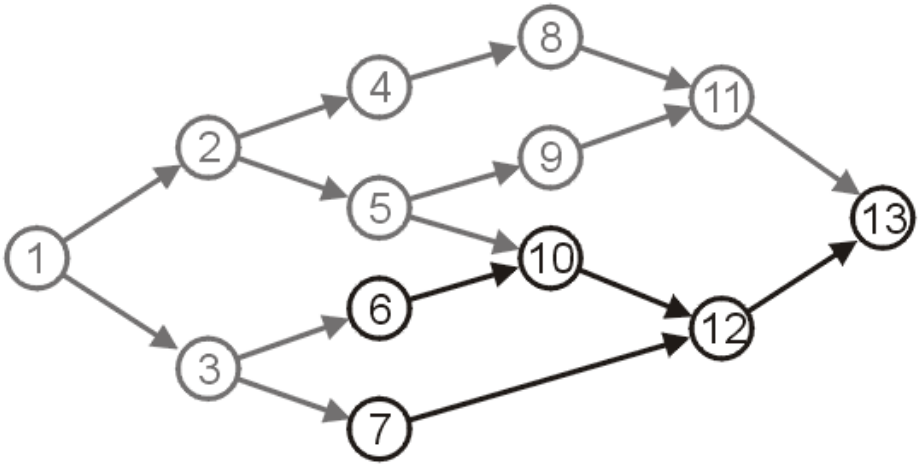
- An example:
  - As we “remove” 1, now 2 and 3 have in-degree 0, so the topological sort could continue with either one of these — we'll choose 2:



1, 2, 4, 8, 5, 9, 11

Topological sort

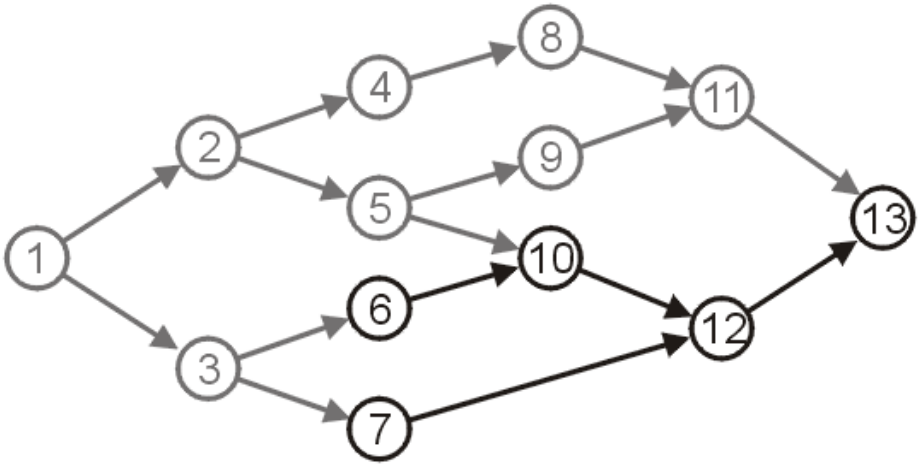
- An example:
  - As we “remove” 1, now 2 and 3 have in-degree 0, so the topological sort could continue with either one of these — we'll choose 2:



1, 2, 4, 8, 5, 9, 11, 3

Topological sort

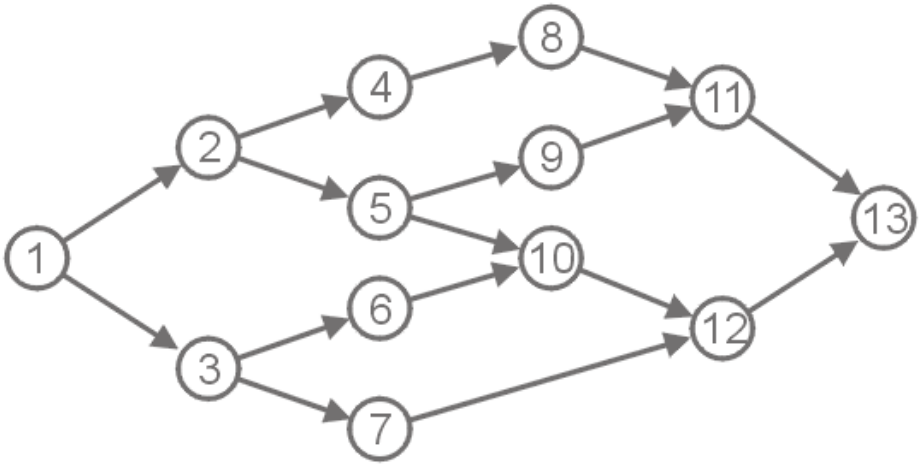
- An example:
  - As we “remove” 1, now 2 and 3 have in-degree 0, so the topological sort could continue with either one of these — we'll choose 2:



1, 2, 4, 8, 5, 9, 11, 3 ··· etc.

Topological sort

- An example:
  - As we “remove” 1, now 2 and 3 have in-degree 0, so the topological sort could continue with either one of these — we'll choose 2:



1, 2, 4, 8, 5, 9, 11, 3, 6, 10, 7, 12, 13

Topological sort

- The fact that a topological sort is not unique should have become quite clear from this example — anyone?

Topological sort

- A typical implementation uses an array of in-degrees, and optionally a queue (for efficiency)

Topological sort

- A typical implementation uses an array of in-degrees, and optionally a queue (for efficiency)
  - We start by initializing the table of in-degrees (how do we do this efficiently? A single pass through the list of vertices, perhaps?)

Topological sort

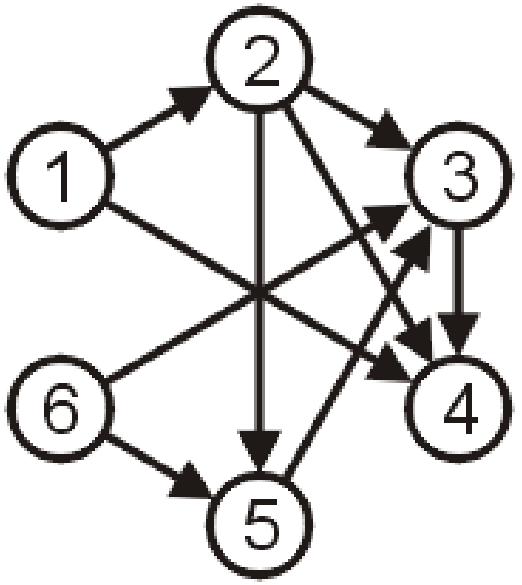
- A typical implementation uses an array of in-degrees, and optionally a queue (for efficiency)
  - We start by initializing the table of in-degrees (how do we do this efficiently? A single pass through the list of vertices, perhaps?)
    - And BTW, why the big deal with doing this efficiently?? How would we do it inefficiently?

Topological sort

- Let's look at an example, directly from Prof. Harder's slides.

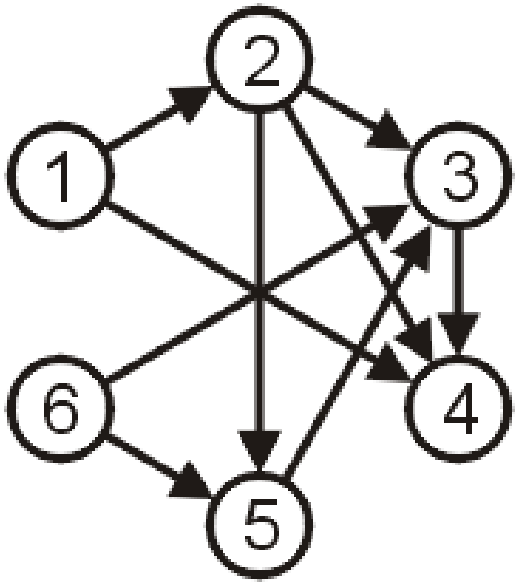
Example

Consider the following DAG with six vertices



Example

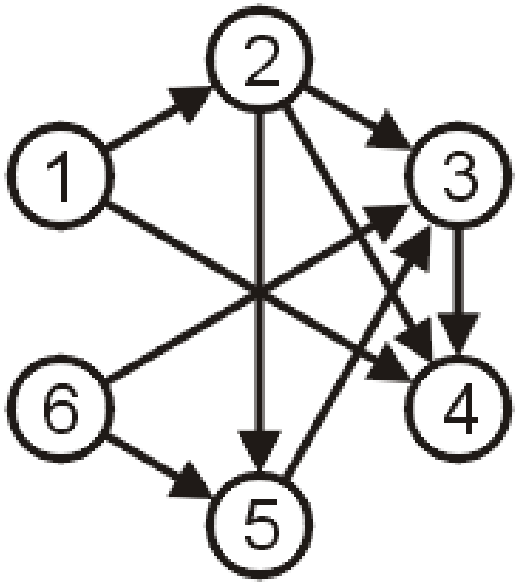
Let us define the table of in-degrees
(or more likely, copy it)



Vertex	In-degree
1	0
2	1
3	3
4	3
5	2
6	0

Example

And a queue into which we can insert vertices 1 and 6



Vertex	In-degree
1	**0**
2	1
3	3
4	3
5	2
6	**0**

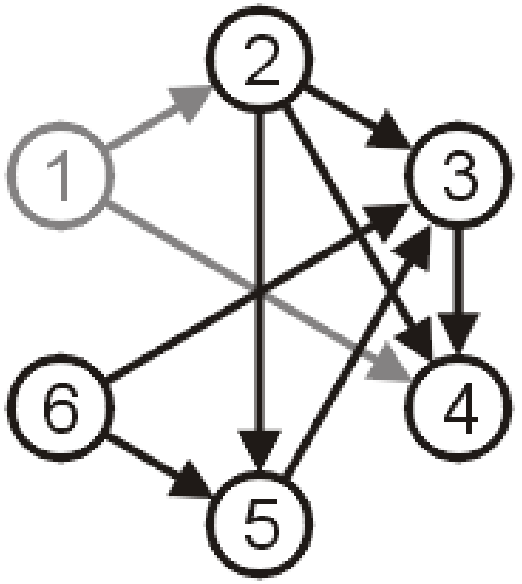
Queue

1 **6**

Example

We dequeue the head (1), decrement the in-degree of all adjacent vertices: 2 and 4

– 2 is decremented to zero: enqueue 2



Vertex	In-degree
1	0
2	**0**
3	3
4	**2**
5	2
6	0

Queue

6 2

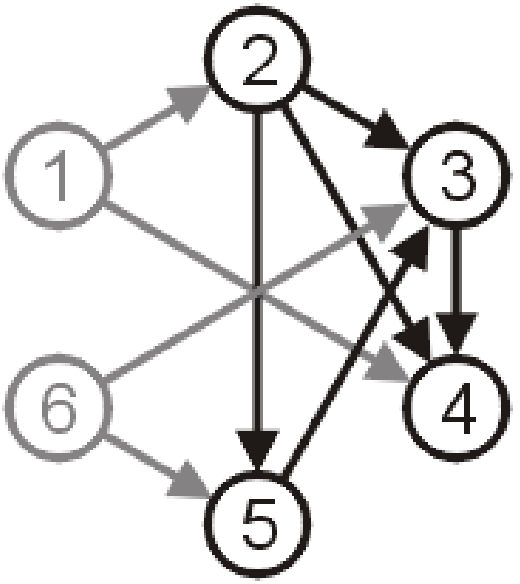
Sort

1

Example

We dequeue 6 and decrement the in-degree of all adjacent vertices

- Neither is decremented to zero



Vertex	In-degree
1	0
2	0
3	**2**
4	2
5	**1**
6	0

Queue

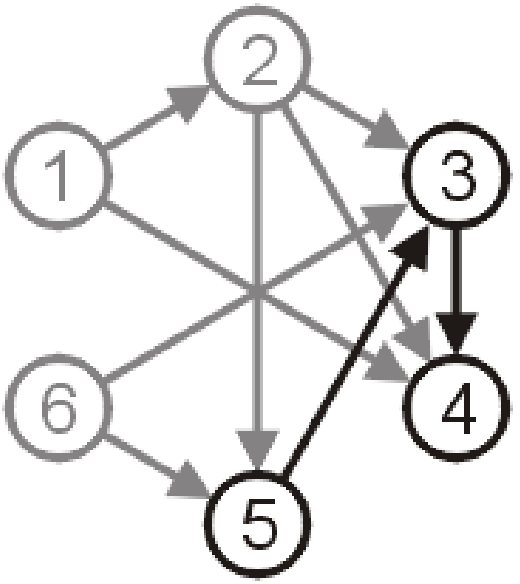
2

Sort

1, 6

Example

We dequeue 2, decrement, and enqueue vertex 5



Vertex	In-degree
1	0
2	0
3	**1**
4	**1**
5	**0**
6	0

Queue

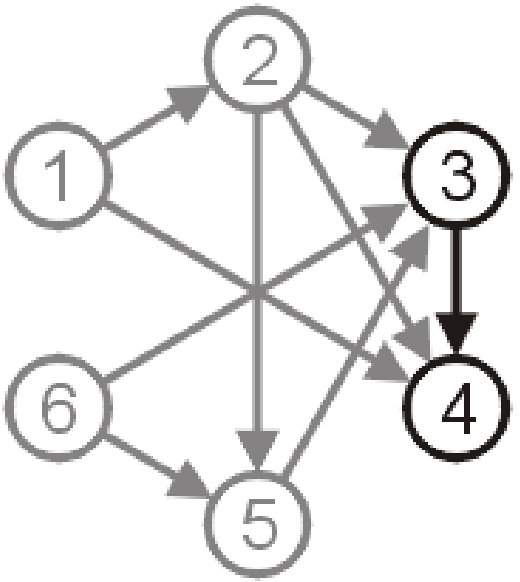
5

Sort

1, 6, 2

Example

We dequeue 5, decrement, and enqueue vertex 3



Vertex	In-degree
1	0
2	0
3	**0**
4	1
5	0
6	0

Queue

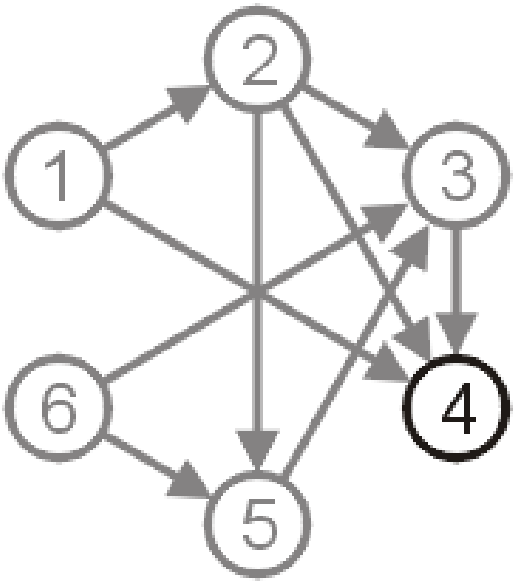
3

Sort

1, 6, 2, 5

Example

We dequeue 3, decrement 4, and add 4 to the queue



Vertex	In-degree
1	0
2	0
3	0
4	**0**
5	0
6	0

Queue

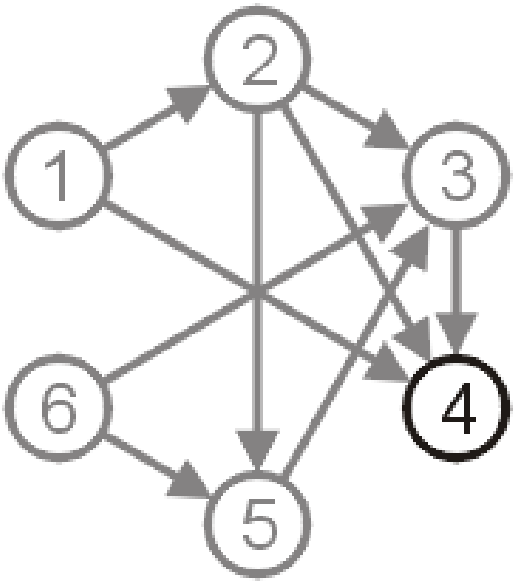
4

Sort

1, 6, 2, 5, 3

Example

We dequeue 4, there are no adjacent vertices to decrement the in degree



Vertex	In-degree
1	0
2	0
3	0
4	0
5	0
6	0

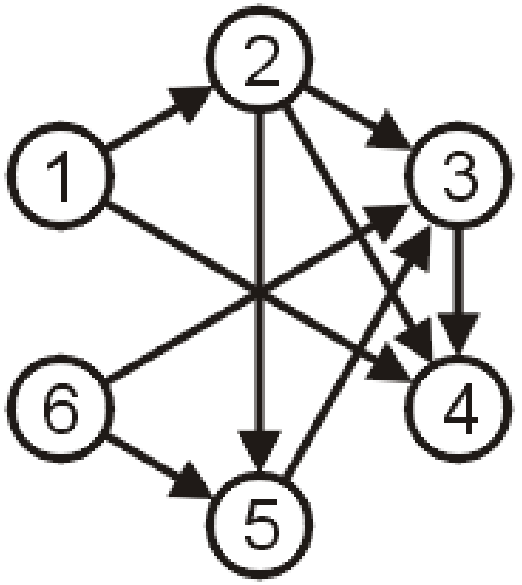
Queue

Sort
1, 6, 2, 5, 3, 4

Example

The queue is now empty, so a topological sort is

1, 6, 2, 5, 3, 4



Summary

- During today's class, we:
  - Looked at Topological sort for Directed Acyclic Graphs (DAGs)
  - Presented an algorithm to implement this operation.
  - Saw some of its applications.