DAGs and topological sort

	Miles Jones	MTThF 8:30-9:50am	CSE 4140

August 16, 2016

Today's plan

1. Definition of DAG.

1. Ordering algorithm on a DAG.

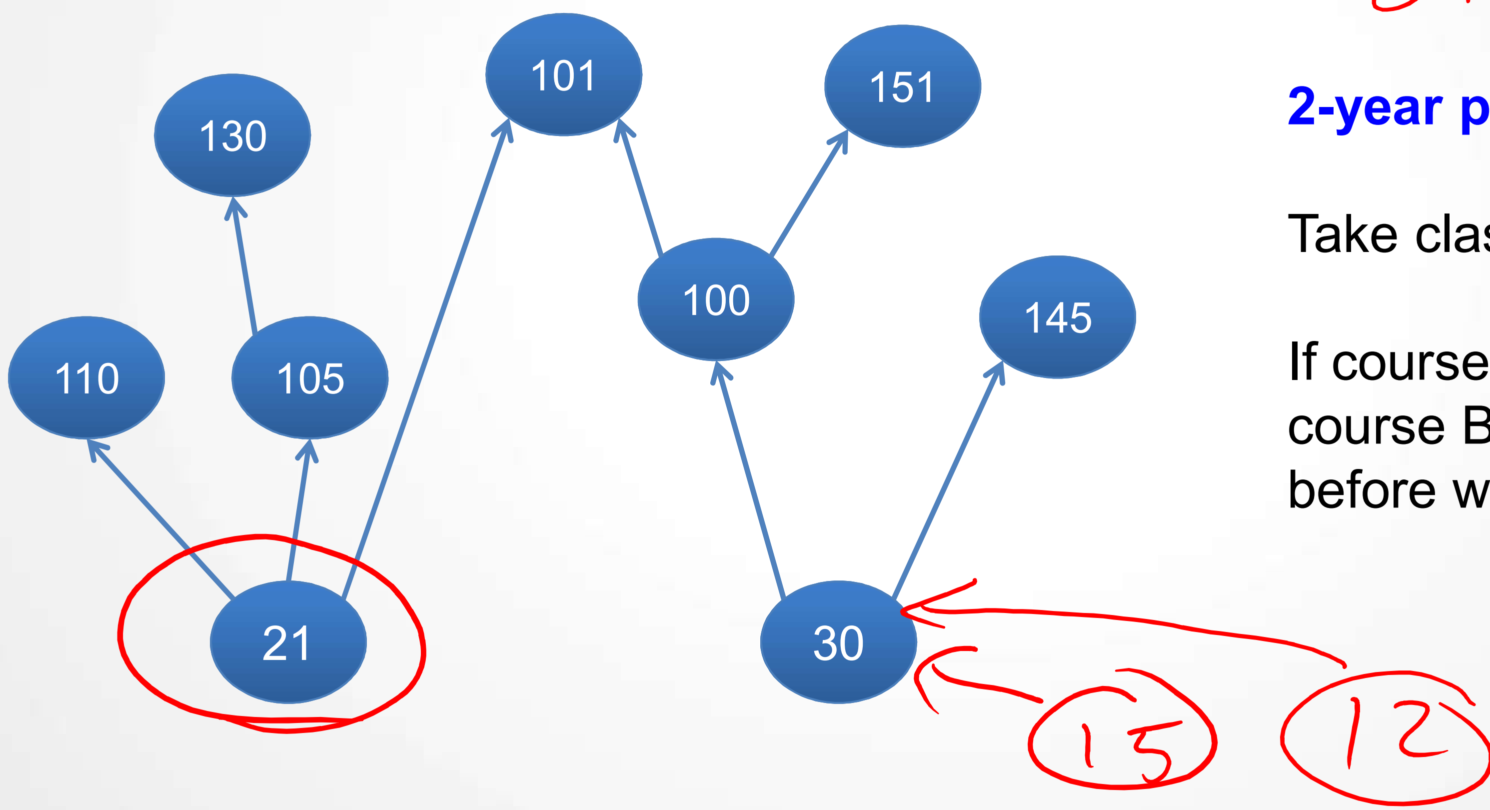
2. Graph search and reachability.

D: Directed
A: Acyclic
G: Graph

In the textbook: Sections 10.4 and 10.5

(some) Prerequisites for (some) CSE classes

A → B if A is prereq of B.

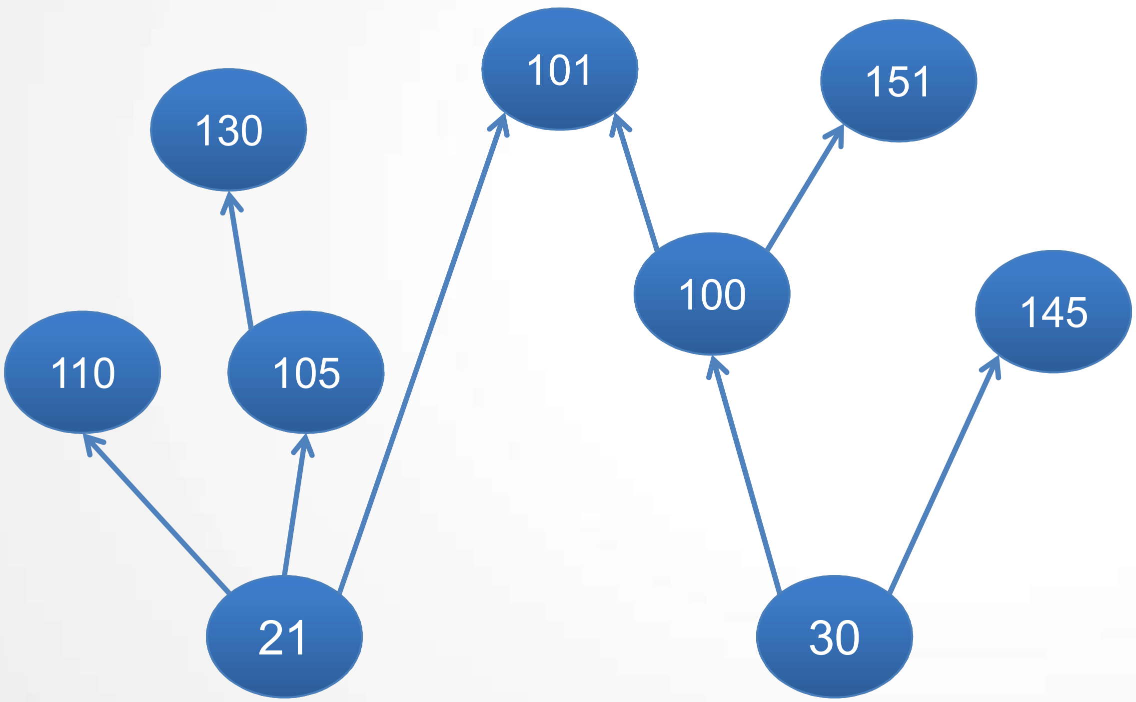


2-year plan:

Take classes in some order.

If course A is prerequisite for course B, must take course A before we take course B.

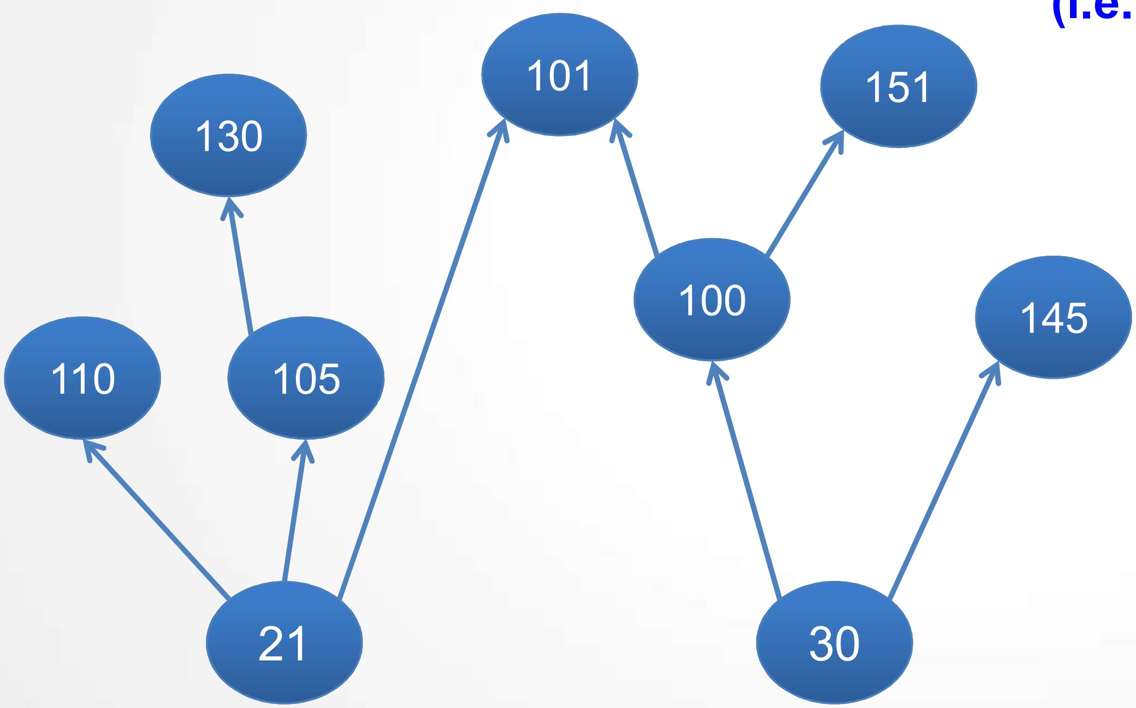
(some) Prerequisites for (some) CSE classes



Which of the following orderings are ok?

A. 30, 145, 151, 100.
B. 110, 105, 21, 101.
C. 21, 105, 130.
D. More than one of the above.
E. None of the above.

(some) Prerequisites for (some) CSE classes

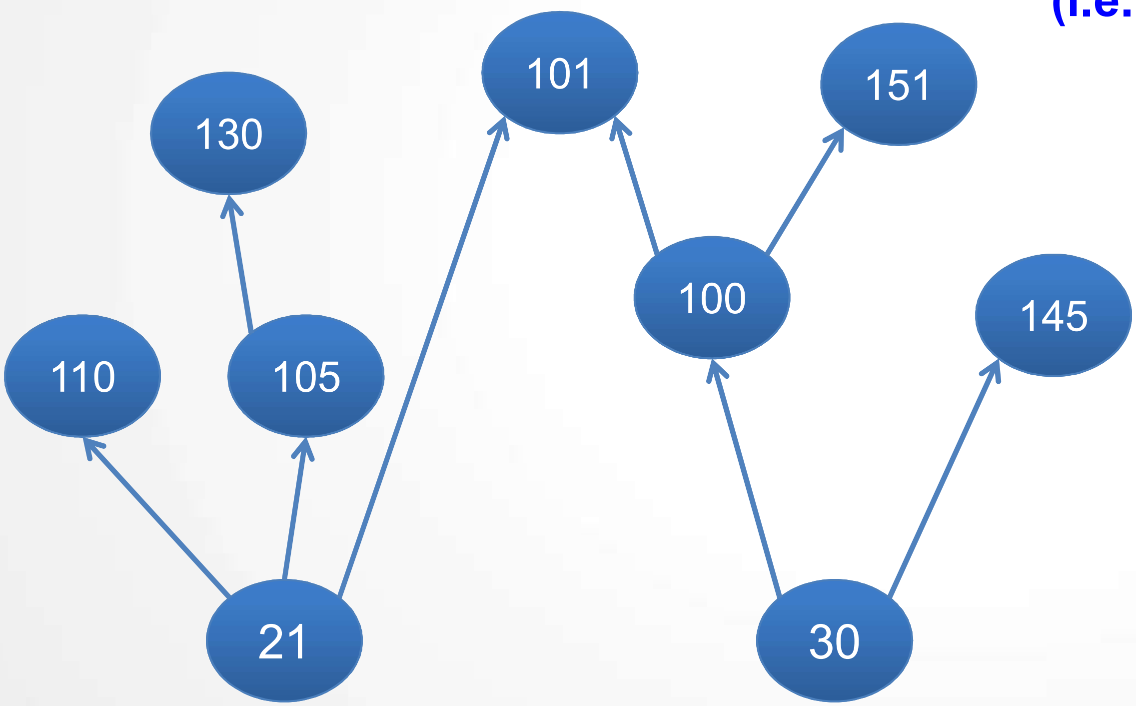


What if we want to include all vertices (i.e. courses) ?

21
110
105
30
130
100
145
101
151

Is this possible for any graph?

(some) Prerequisites for (some) CSE classes



What if we want to include all vertices (i.e. courses) ?

Is this possible for any graph?

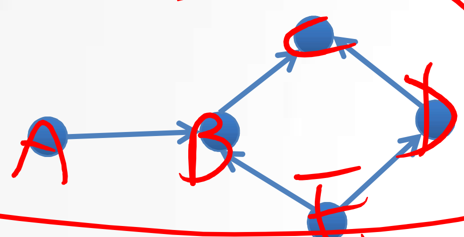
1. **Classify graphs for which it is.**

2. **For those, find a good ordering.**

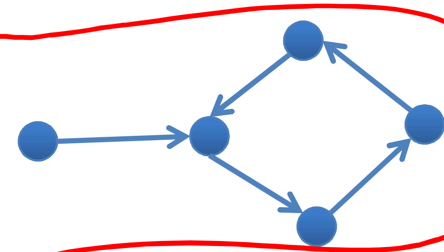
Barriers to ordering

Which of the following graphs have good orderings?

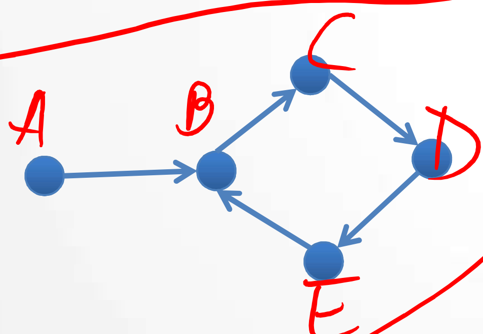
A.



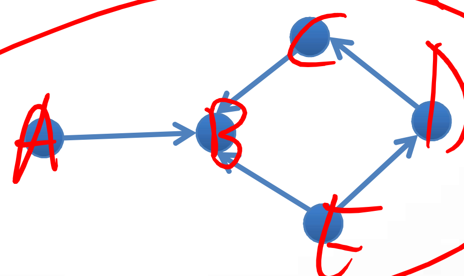
C.



B.



D.



E. None of the above.

B: A $\boxed{B}$ C D E

A: A E B D C

D: A E D C B

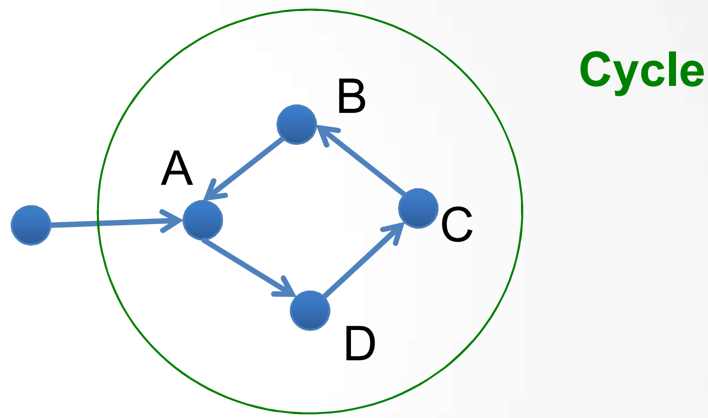
Barriers to ordering

A can't be first (because B is before it).

B can't be first (because C is before it).

C can't be first (because D is before it).

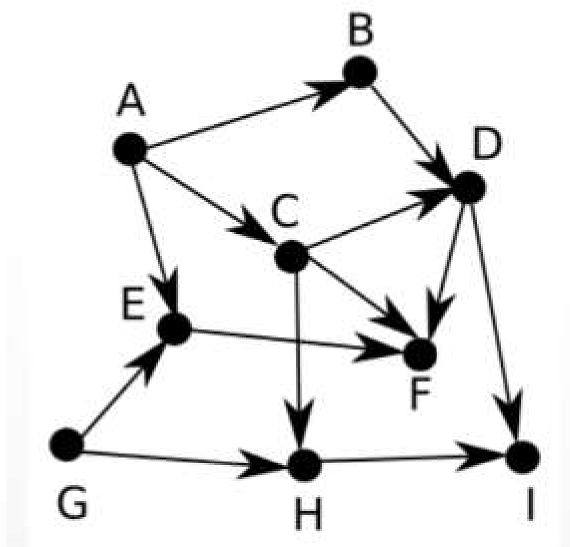
D can't be first (because A is before it).



Whenever there is a cycle, can't find a "good" ordering.

Directed Acyclic Graphs

Directed graphs with no cycles are called **directed acyclic graphs** (DAGs).

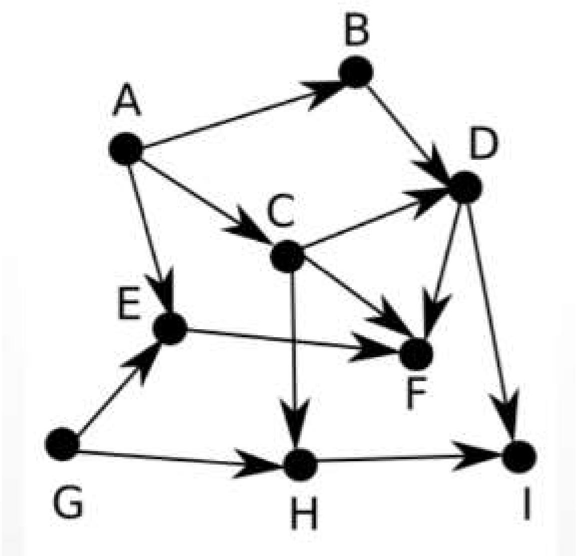


A
G
B
C
E

H
D
F
I

Directed Acyclic Graphs

Directed graphs with no cycles are called **directed acyclic graphs** (DAGs).

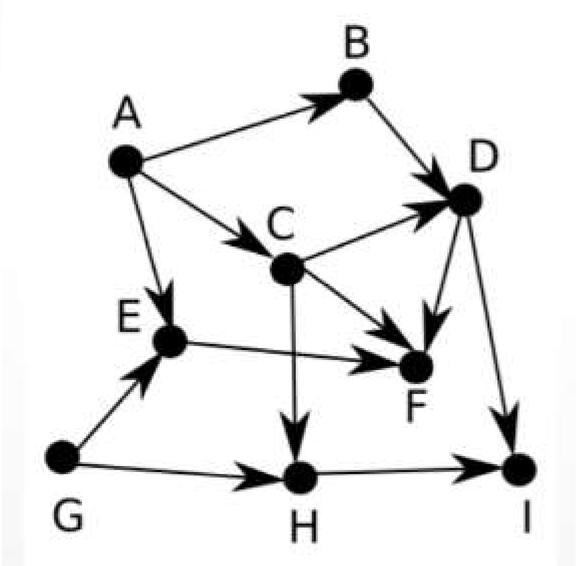


ABGCEHDFI

A **topological ordering** of a graph is an (ordered) list of all its vertices such that, for each edge (v,w) in the graph, v comes before w in the list.

Topological ordering

A: B C E
B: D



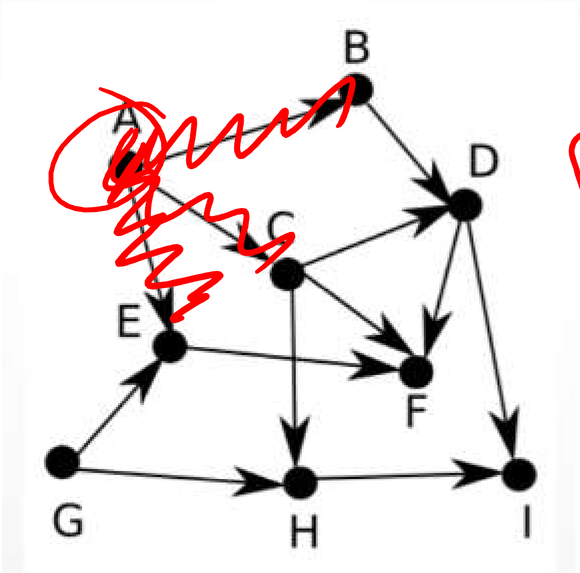
ABGCEHDFI

Two algorithmic questions:

1. Given an (ordered) list of all vertices in the graph, is it a topological ordering?
2. Given a graph, produce a topological ordering.

Topological ordering

A



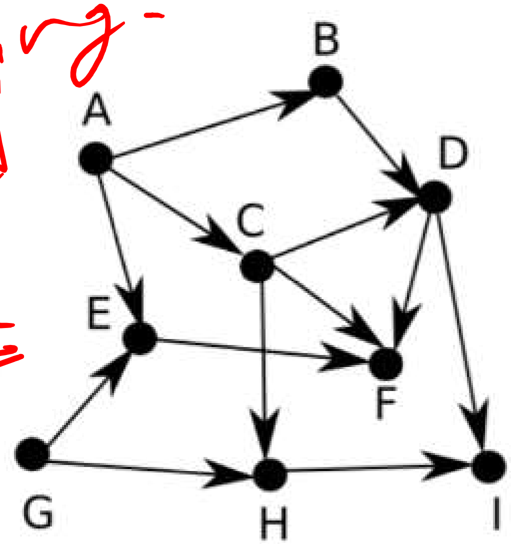
ABGCEHDFI

1. Given an (ordered) list of all vertices in the graph, is it a topological ordering?

How would you do it?

Topological ordering

A vertex with no incoming edges is called a Source



ABGCEHDFI

2. Given a graph, produce a topological ordering.

At what vertex should we start?

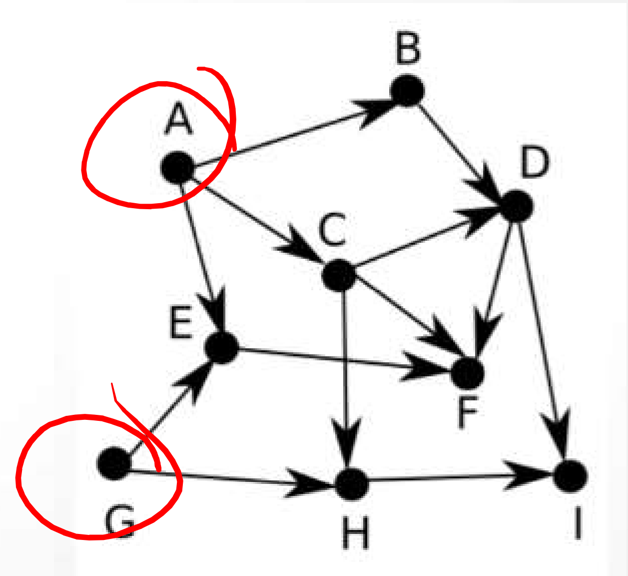
A. Any vertex is okay.
B. We must start at A.
C. Choose any vertex with at least one outgoing edge.
D. Choose any vertex with no incoming edges.
E. None of the above.

Sources of a DAG

In a DAG, vertices with no incoming edges are called **sources**.

Which of these vertices are sources?

A. Only A and G.
B. Only A.
C. Only I.
D. Only I and F.
E. None of the above.



Sources of a DAG

Lemma 1: Every DAG has a (at least one) source

How would you prove this?

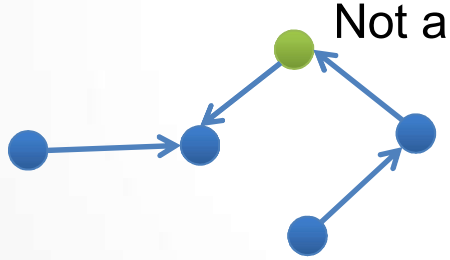
Hint: there are finitely many vertices.

Sources of a DAG

Lemma 1: Every DAG has a (at least one) source

How would you prove this?

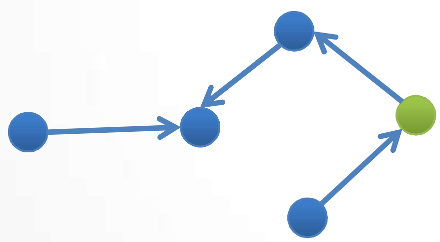
Not a source … look at incoming edges



Sources of a DAG

Lemma 1: Every DAG has a (at least one) source

How would you prove this?

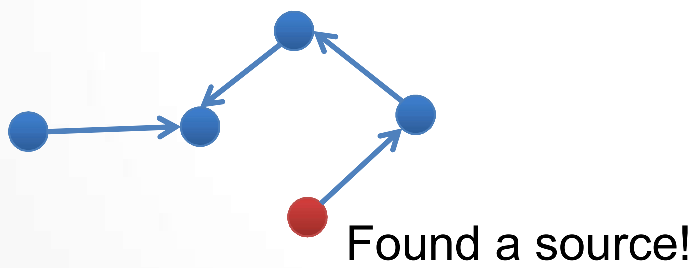


Not a source … look at incoming edges

Sources of a DAG

Lemma 1: Every DAG has a (at least one) source

How would you prove this?



Sources of a DAG

Lemma 1: Every DAG has a (at least one) source

Let G be a DAG. We want to show that G has a source vertex.

In a proof by contradiction (aka indirect proof), what should we assume?

A. G has a source vertex.
B. All the vertices in G are sources.
C. No vertex in G is a source.
D. G has at least one source vertex and at least one vertex that's not a source.
E. None of the above.

Sources of a DAG

Lemma 1: Every DAG has a (at least one) source

Proof of Lemma 1: Let G be a DAG with n (n>1) vertices.
We want to show that G has a source vertex.

Assume towards a contradiction that no vertex in G is a source.

Let v_0 be a vertex in G. Since v is not a source (by assumption), it has an incoming edge. Let v_1 be a vertex in G so that (v_1, v_0) is an edge in G. Since v_1 is also not a source, let v_2 be a vertex in G so that (v_2, v_1) is an edge in G. Keep going to find

$$v_0, v_1, v_2, \ldots, v_n$$

vertices. There must be a **repeated** vertex in this list (Pigeonhole Principle).
Contradiction with G being acyclic.

Sources of a DAG

Notation: **G-v** is the graph that results when remove v and all of its incident edges from G.

Lemma 2: If v is a **source vertex** of G, then G is a DAG if and only if G-v is a DAG.

Sources of a DAG

Notation: **G-v** is the graph that results when remove v and all of its incident edges from G.

Lemma 2: If v is a **source vertex** of G, then
G is a DAG if and only if G-v is a DAG.

Proof of Lemma 2: Let G be a DAG and assume v is a vertex in G.

Assume G is a DAG. WTS G-v is a DAG.

Assume G-v is a DAG. WTS G is a DAG.

Sources of a DAG

Notation: **G-v** is the graph that results when remove v and all of its incident edges from G.

Lemma 2: If v is a **source vertex** of G, then G is a DAG if and only if G-v is a DAG

Proof of Lemma 2: Let G be a DAG and assume v is a vertex in G.

Assume G is a DAG. WTS G-v is a DAG. Can't introduce any cycles by removing edges.

Assume G-v is a DAG. WTS G is a DAG.

Sources of a DAG

Notation: **G-v** is the graph that results when remove v and all of its incident edges from G.

Lemma 2: If v is a **source vertex** of G, then G is a DAG if and only if G-v is a DAG

Proof of Lemma 2: Let G be a DAG and assume v is a vertex in G.

Assume G is a DAG. WTS G-v is a DAG. Can't introduce any cycles by removing edges.

Assume G-v is a DAG. WTS G is a DAG. ?? Contrapositive …

Assume G is not a DAG. Show G-v is not a DAG.

Sources of a DAG

Notation: **G-v** is the graph that results when remove v and all of its incident edges from G.

Lemma 2: If v is a **source vertex** of G, then
G is a DAG if and only if G-v is a DAG

Proof of Lemma 2: Let G be a DAG and assume v is a vertex in G.

Assume G is a DAG. WTS G-v is a DAG. Can't introduce any cycles by removing edges.

Assume G is not a DAG. WTS G-v is not a DAG. A cycle in G can't include a source (because no incoming edges). So this cycle will also be in G-v.

Find Topological Ordering (if possible)

G is DAG → G has source

G has no source → G is not a DAG

While G has at least one vertex

 If G has some source,

  Choose one source and output it.

  Delete the source and all its outgoing edges from G.

 Else

  Return that G is not a DAG.

Graph comes as an adjacency list

Find Topological Ordering (if possible)

While G has at least one vertex
    If G has some source,
        Choose one source and output it.
        Delete the source and all its outgoing edges from G.
    Else
        Return that G is not a DAG.

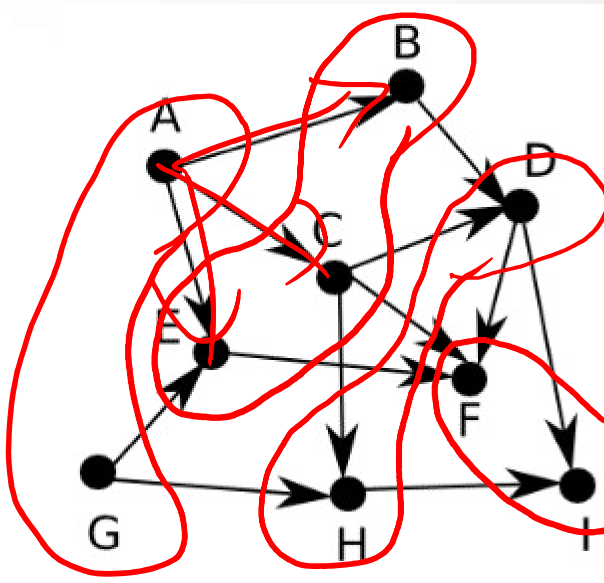
Implementation details:
Maintain integer array, **InDegree[]**, of length n
Maintain collection of sources, **S**, as list, stack, or queue.

```
Choose first x in S.
For each y adjacent to x,
    Decrement InDegree[y]
    If InDegree[y]=0, add y to S.
```

Example

InDegree[]

A	B	C	D	E	F	G	H	I
0	1 0	1 0	2	2 1	3	0	2	2



Collection of sources: **S** = A, G

Output:

A

A G B C

```
Choose first x in S.
For each y adjacent to x,
    Decrement InDegree[y]
    If InDegree[y]=0, add y to S.
```

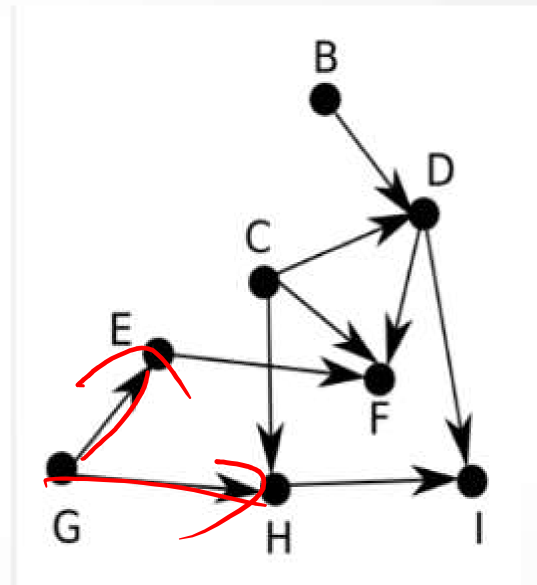
Example

InDegree[]

~~A~~	B	C	D	E	F	G	H	I
~~0~~	0	0	2	~~1~~ 0	3	0	~~2~~ 1	2

Collection of sources: **S** = G, B, C E

Output: A G




```
Choose first x in S.
For each y adjacent to x,
    Decrement InDegree[y]
    If InDegree[y]=0, add y to S.
```

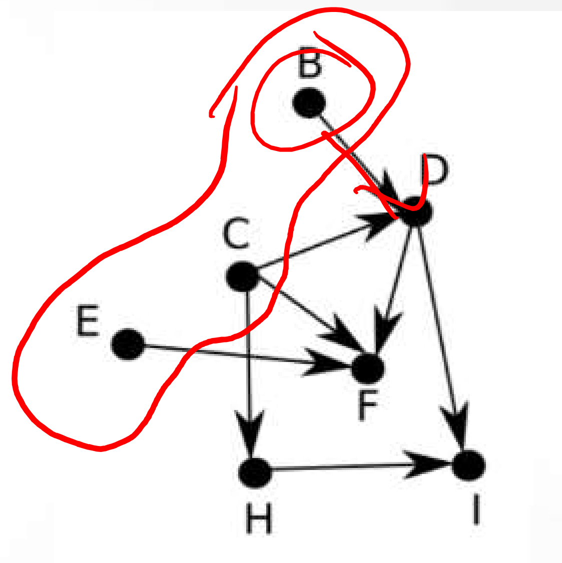
Example

InDegree[]

~~A~~	B	C	D	E	F	~~G~~	H	I
~~0~~	0	0	~~2~~ 1	0	3	~~0~~	1	2

Collection of sources: **S** = B, C, E

Output: A, G




```
Choose first x in S.
For each y adjacent to x,
    Decrement InDegree[y]
    If InDegree[y]=0, add y to S.
```

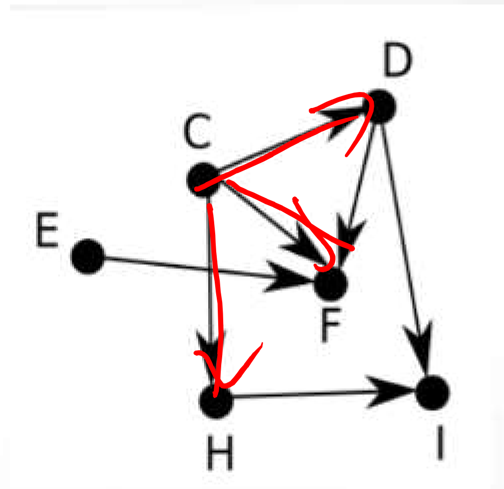
Example

InDegree[]

~~A~~	~~B~~	C	D	E	F	~~G~~	H	I
~~0~~	~~0~~	0	~~1~~ 0	0	~~3~~ 2	~~0~~	~~1~~ 0	2

Collection of sources: **S** = C, E

Output: A, G, B




```
Choose first x in S.
For each y adjacent to x,
    Decrement InDegree[y]
    If InDegree[y]=0, add y to S.
```

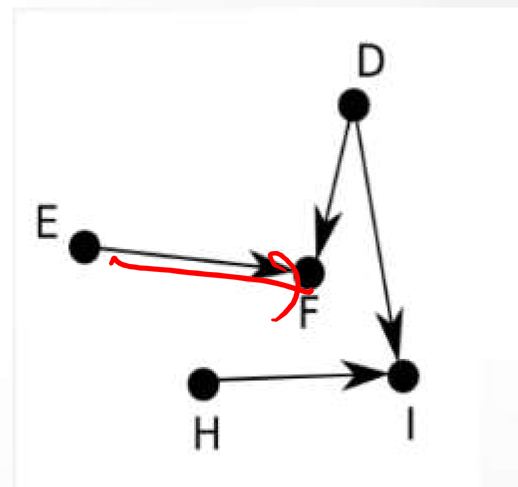
Example

InDegree[]

~~A~~	~~B~~	~~C~~	D	E	F	~~G~~	H	I
~~0~~	~~0~~	~~0~~	0	0	~~2~~ 1	~~0~~	0	2

Collection of sources: **S** = E, D, H

Output: A, G, B, C




```
Choose first x in S.
For each y adjacent to x,
    Decrement InDegree[y]
    If InDegree[y]=0, add y to S.
```

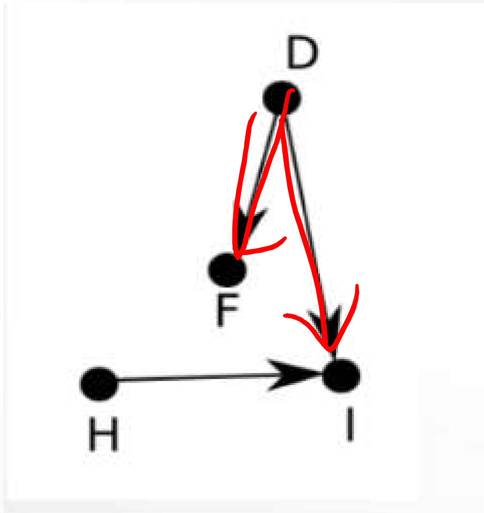
Example

InDegree[]

~~A~~	~~B~~	~~C~~	D	~~E~~	F	~~G~~	H	I
~~0~~	~~0~~	~~0~~	0	~~0~~	~~1~~ 0	~~0~~	0	~~2~~ 1

Collection of sources: **S** = D, H F

Output: A, G, B, C, E D




```
Choose first x in S.
For each y adjacent to x,
    Decrement InDegree[y]
    If InDegree[y]=0, add y to S.
```

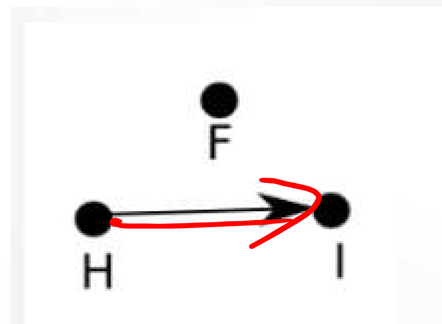
Example

InDegree[]

~~A~~	~~B~~	~~C~~	~~D~~	~~E~~	F	~~G~~	H	I
~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	0	~~0~~	0	~~1~~ 0

Collection of sources: **S** = H, F

Output: A, G, B, C, E, D




```
Choose first x in S.
For each y adjacent to x,
    Decrement InDegree[y]
    If InDegree[y]=0, add y to S.
```

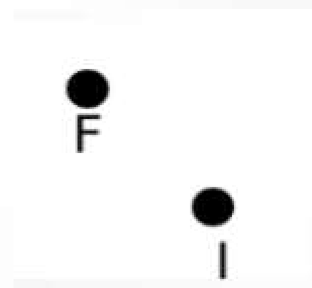
Example

InDegree[]

~~A~~	~~B~~	~~C~~	~~D~~	~~E~~	F	~~G~~	~~H~~	I
~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	0	~~0~~	~~0~~	0

Collection of sources: **S** = F, I

Output: A, G, B, C, E, D, H




```
Choose first x in S.
For each y adjacent to x,
    Decrement InDegree[y]
    If InDegree[y]=0, add y to S.
```

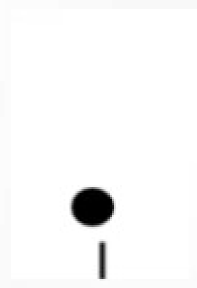
Example

InDegree[]

~~A~~	~~B~~	~~C~~	~~D~~	~~E~~	F	~~G~~	~~H~~	I
~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	0	~~0~~	~~0~~	0

Collection of sources: **S** = I

Output: A, G, B, C, E, D, H, F




```
Choose first x in S.
For each y adjacent to x,
    Decrement InDegree[y]
    If InDegree[y]=0, add y to S.
```

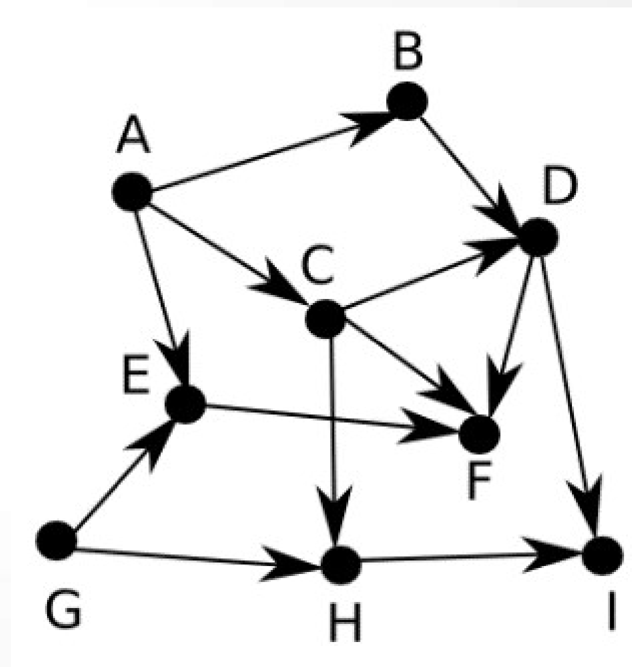
Example

InDegree[]

~~A~~	~~B~~	~~C~~	~~D~~	~~E~~	F	~~G~~	~~H~~	I
~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	0	~~0~~	~~0~~	0

Collection of sources: **S** =

Output: A, G, B, C, E, D, H, F, I




```
Choose first x in S.
For each y adjacent to x,
    Decrement InDegree[y]
    If InDegree[y]=0, add y to S.
```

Find Topological Ordering (if possible)

Make an array of indegree[v] for each v in |V|
Initialize a queue of sources S

$O(|V|+|E|)$

While G has at least one vertex

Goes through every vertex

If G has some source, If S is not empty then

$O(1)$

Choose one source and output it. v:= head(S), output v, eject(S,v)

Delete the source and all its outgoing edges from G.

for each (v,u) in E
decrement indegree(u)
if indegree(u)=0 then insert (S,u)

for vertex v

$O(\text{outdegree}(v))$

Else

Return that G is not a DAG.

Find Topological Ordering (if possible)

Make an array of indegree[v] for each v in |V|
Initialize a queue of sources S

O(|V|+|E|)

While G has at least one vertex

If G has some source, If S is not empty then O(1)

Choose one source and output it. v:= head(S), output v, eject(S,v) O(1)

Delete the source and all its outgoing edges from G.

for each (v,u) in E

decrement indegree(u)

if indegree(u)=0 then insert (S,u)

out

O(degree(v))

Else

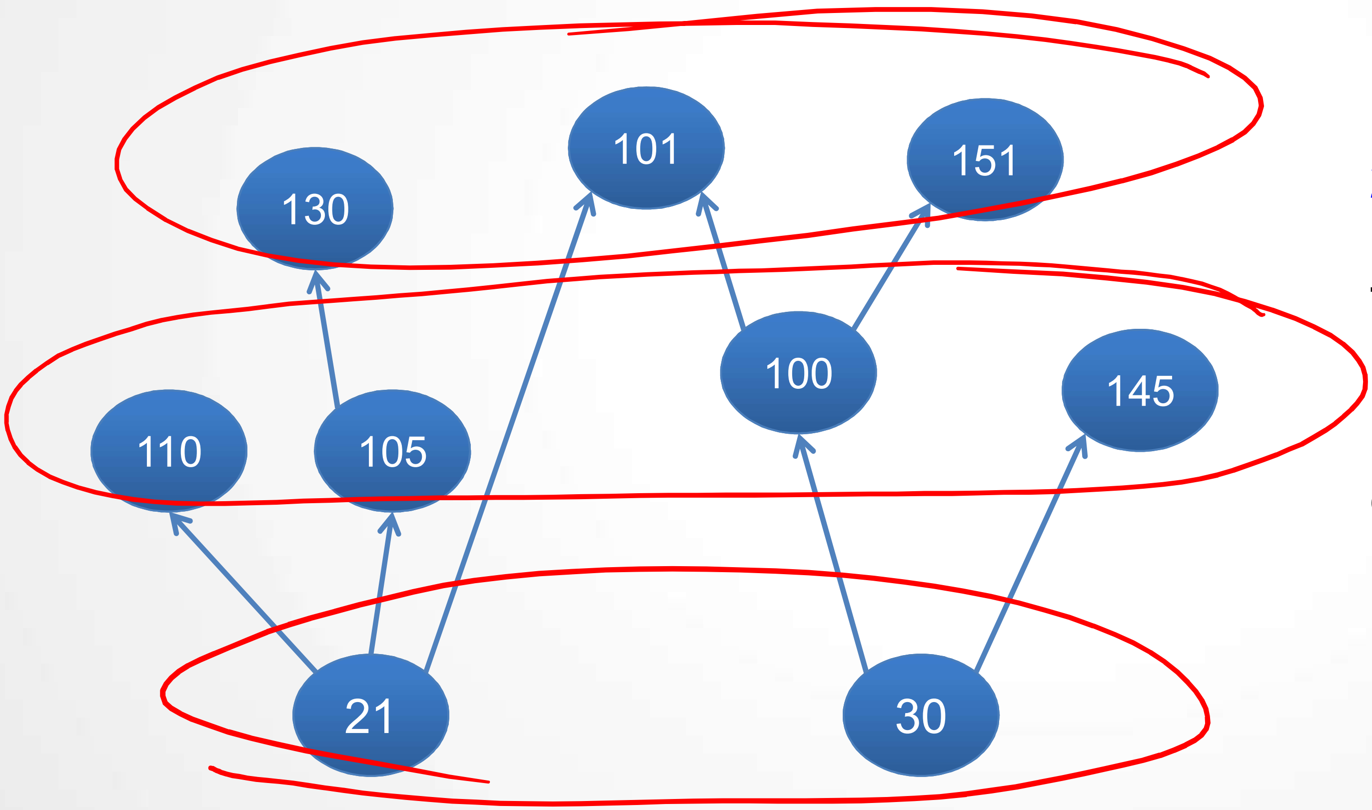
Return that G is not a DAG.

$O(|V|+|E|)$

Since each v is ejected once, total time is

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

(some) Prerequisites for (some) CSE classes



2-year plan:

Take classes in some order.

If course A is prerequisite for course B, must take course A before we take course B.

How many quarters?

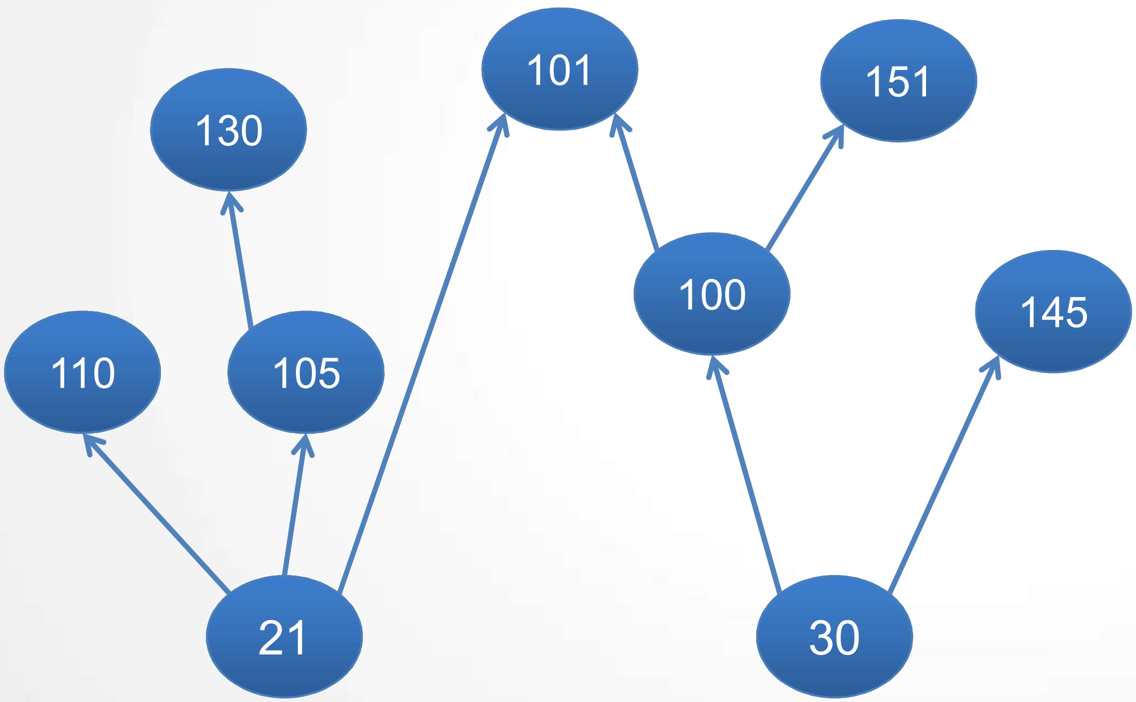
Layers of a DAG

First layer **all nodes that are sources**

Next layer **all nodes that are now sources**
(once we remove previous layer and its outgoing edges)

Repeat…

(some) Prerequisites for (some) CSE classes



How many quarters (layers) before take all classes?

A. 1.
B. 2.
C. 3.
D. 4.
E. More than four.