

Directed graphs & topological ordering

Reading: Kleinberg & Tardos

Ch. 3.5 and 3.6

Additional resource: CLRS Ch. 22

Recall...

What do these have in common?

You have embarked on...

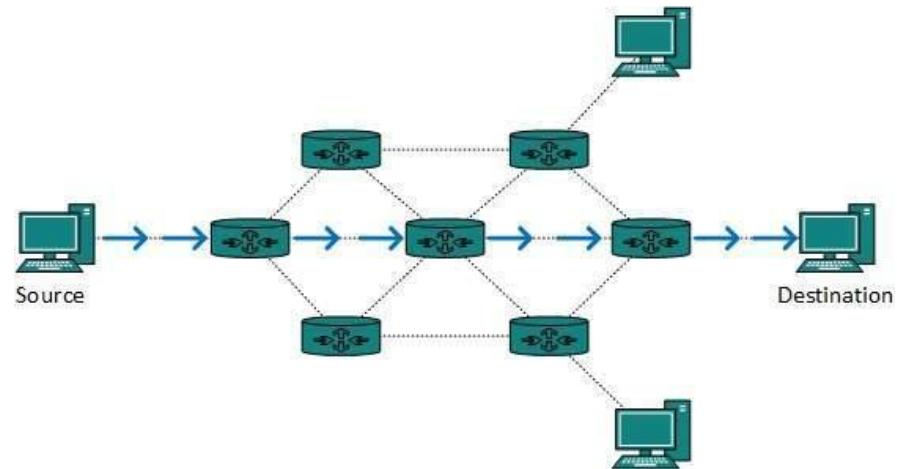
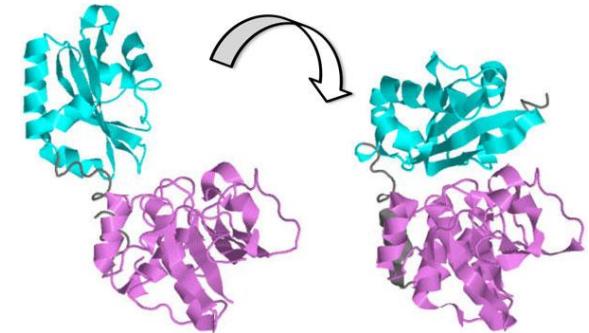
Trebuchet Tales

Soldier Encounter

You see some angry soldiers on a faraway hill.

Launch a 95 kilogram stone projectile over 300 meters. >

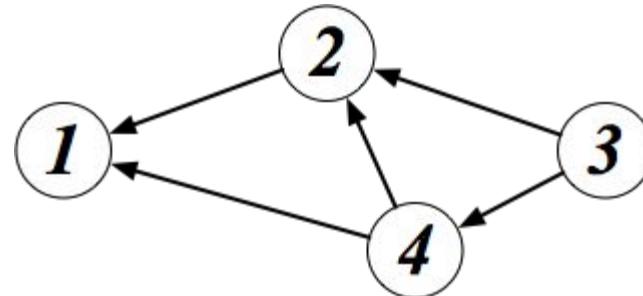
Do nothing. >



Directed graphs

Directed graph: definitions and terminology

- A **graph** $G = (V, E)$
 - V is a set of **vertices**
 - $E \subseteq V \times V$ is the set of **edges**
- **directed edge** $e = (u, v)$
 - u : **source**
 - v : **target**
 - **outgoing edge** from u
 - **Incoming edge** to v

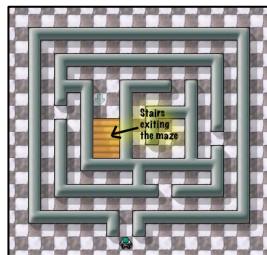
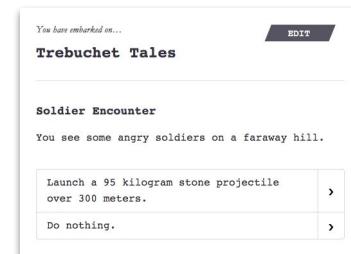
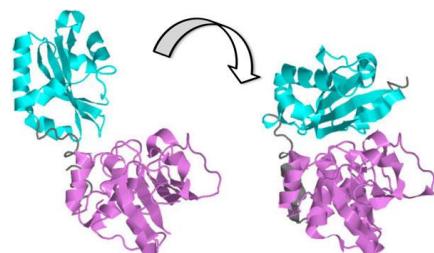


Directed graphs: examples

undirected



directed



Directed graph: paths

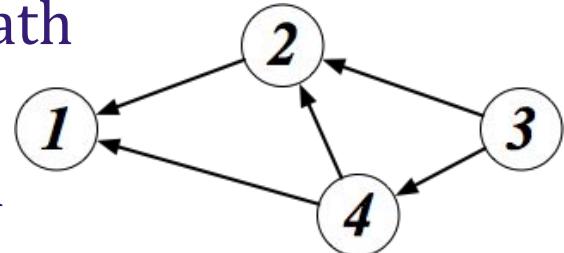
- (Directed) path, cycle: same as for undirected

$v_1, v_2, \dots, v_{k-1}, v_k$

- What about connectivity?

- Hmm... $s \rightarrow t$ path doesn't mean $t \rightarrow s$ path

- t is **reachable** from s if there is a $s \rightarrow t$ path
- How to find all nodes reachable from u ?
 - Example: all words that start with letter "a"
- What is the length of the shortest directed path from s to t ?



Directed graph traversals

- DFS/BFS: same as for undirected
 - “neighbors”: outgoing neighbors
- Adjacency lists
 - Outgoing neighbors
 - Incoming neighbors
- Problem: Find all nodes s with $s \rightarrow t$ path?
- Algorithm: BFS following edges in reverse direction.

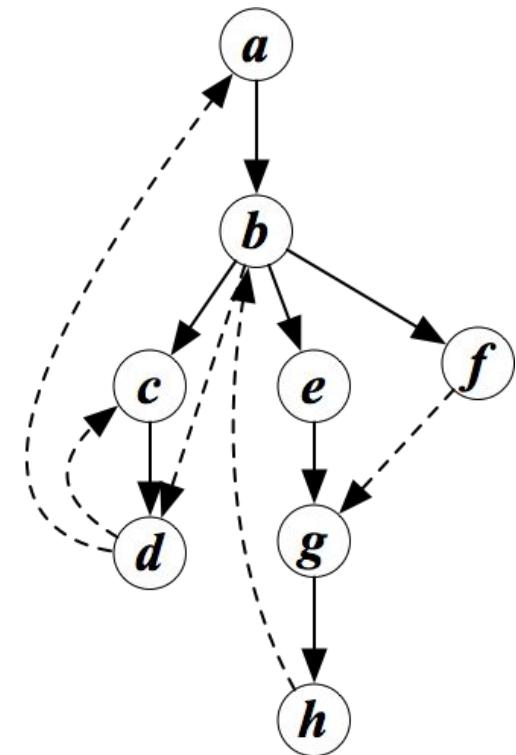
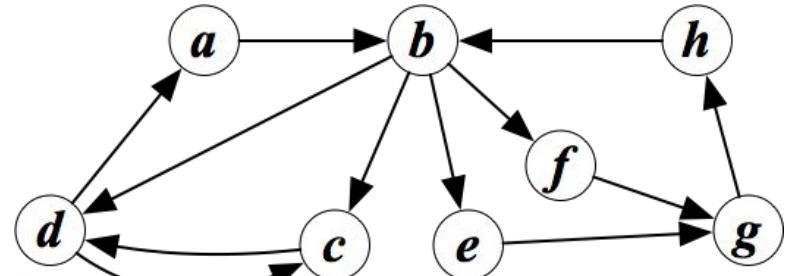
DFS on a directed graph

DFS(G)

```
1  for each vertex  $u \in G.V$ 
2       $u.color = \text{WHITE}$ 
3       $u.\pi = \text{NIL}$ 
4   $time = 0$ 
5  for each vertex  $u \in G.V$ 
6      if  $u.color == \text{WHITE}$ 
7          DFS-VISIT( $G, u$ )
```

DFS-VISIT(G, u)

```
1   $time = time + 1$                                 // white vertex  $u$  has just been discovered
2   $u.d = time$ 
3   $u.color = \text{GRAY}$ 
4  for each  $v \in G.Adj[u]$ 
5      if  $v.color == \text{WHITE}$ 
6           $v.\pi = u$ 
7          DFS-VISIT( $G, v$ )
8   $u.color = \text{BLACK}$                                 // blacken  $u$ ; it is finished
9   $time = time + 1$ 
10  $u.f = time$ 
```



BFS on a directed graph

BFS(G, v):

for each vertex u in $G.V$

$u.visited = \text{false}$

$v.layer = 0$

$v.parent = \text{null}$

$Q = \text{empty queue}$

$Q.enqueue(v)$

 while $\text{!}Q.\text{empty}()$

$x = Q.dequeue()$

 for each neighbor y of x

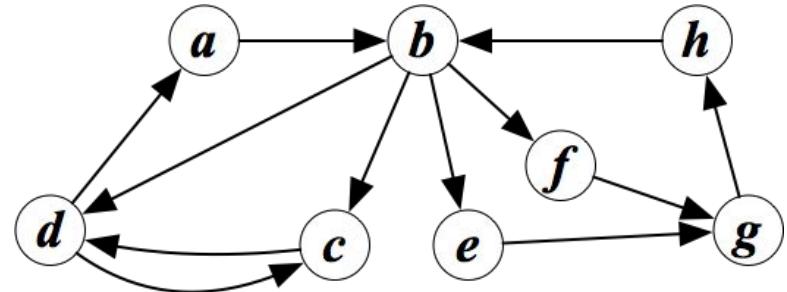
 if $\text{!}(y.visited)$

$y.visited = \text{true}$

$y.parent = x$

$y.layer = x.layer + 1$

$Q.enqueue(y)$



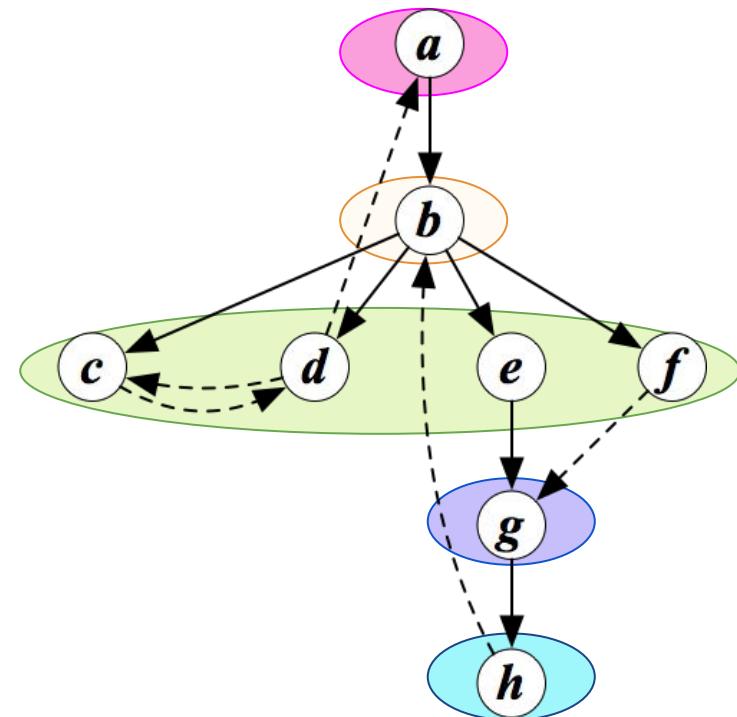
layer 0

layer 1

layer 2

layer 3

layer 4



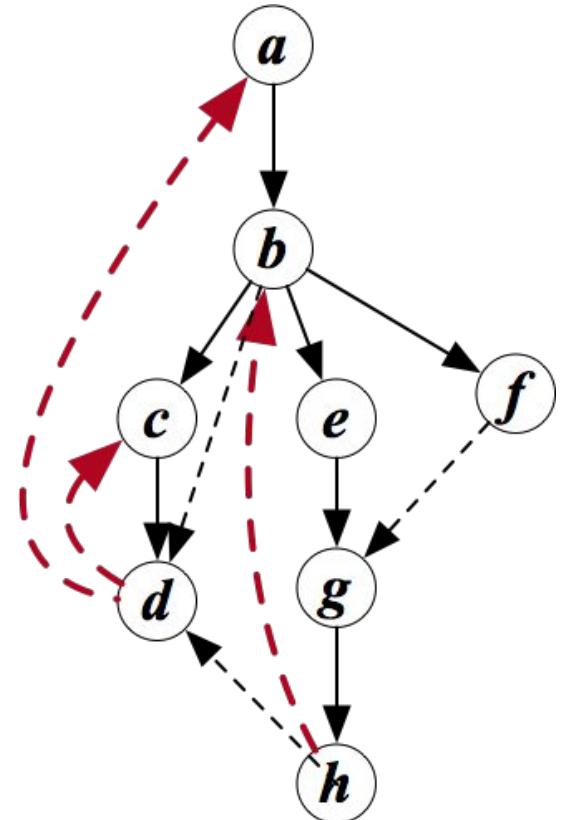
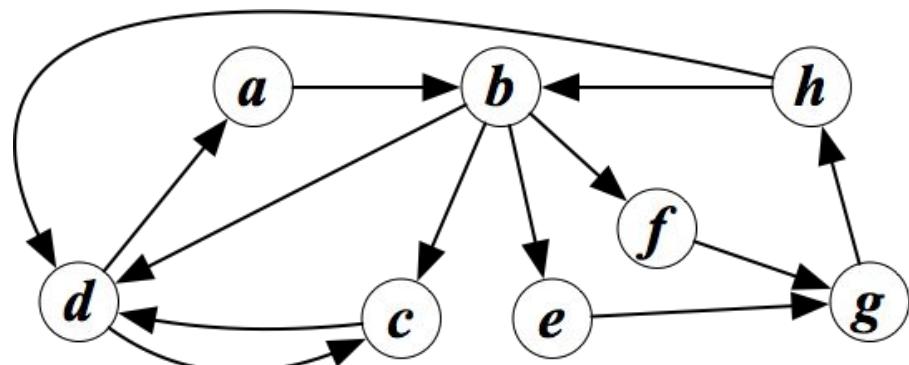
[CLRS]

CS 312 - Algorithms - Mount Holyoke College

Directed DFS: non-tree edges

3 types:

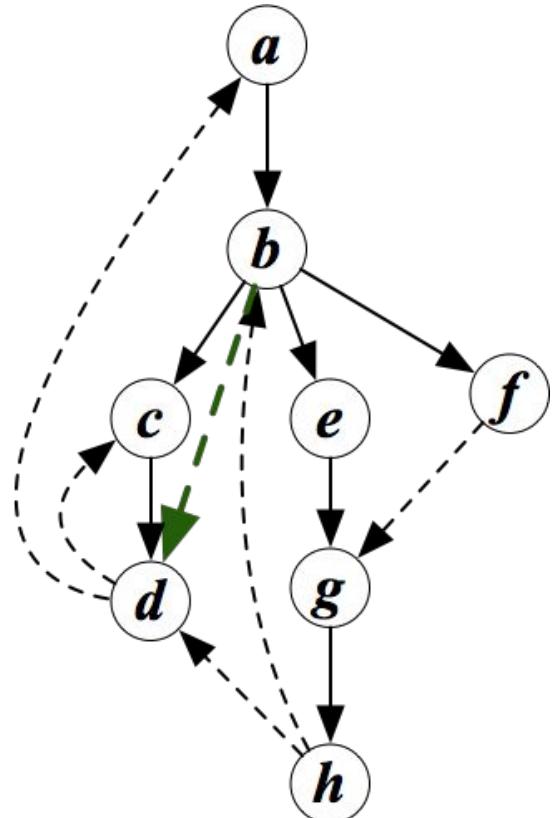
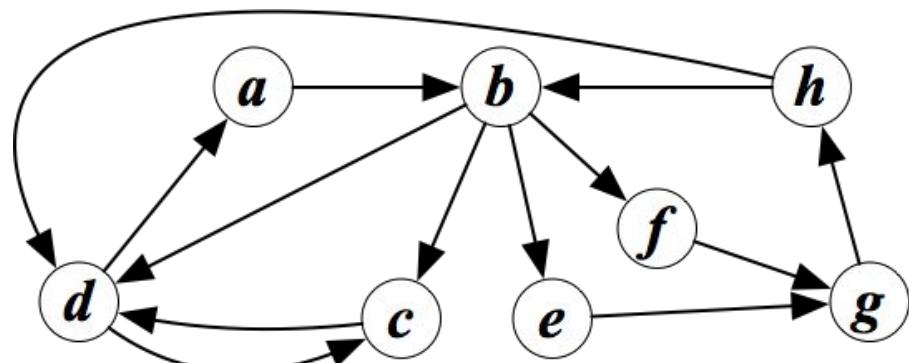
- **Back:** to an ancestor



Directed DFS: non-tree edges

3 types:

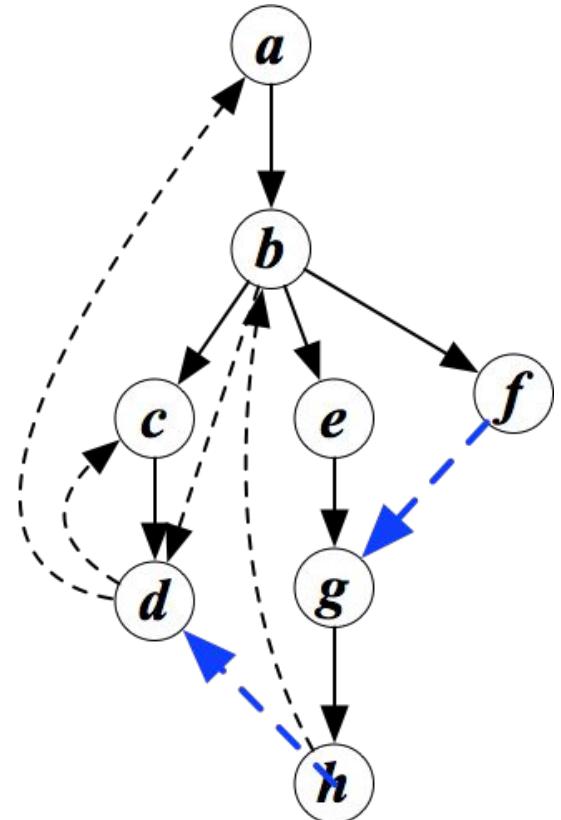
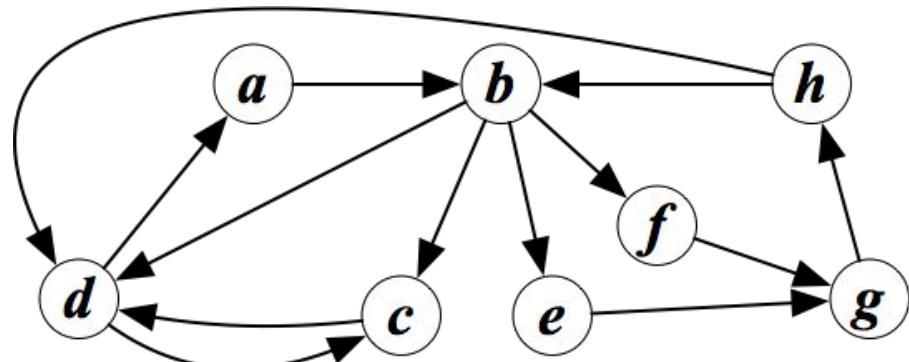
- **Back**: to an ancestor
- **Forward**: to a descendant



Directed DFS: non-tree edges

3 types:

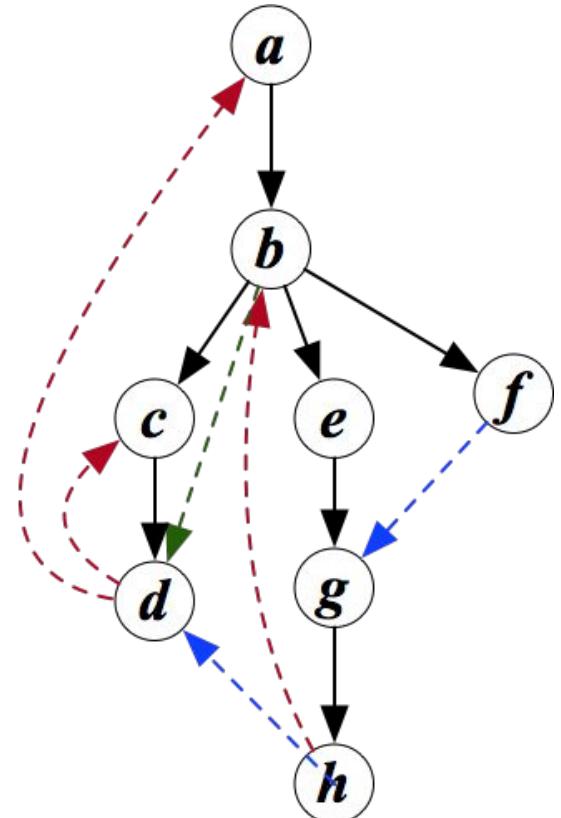
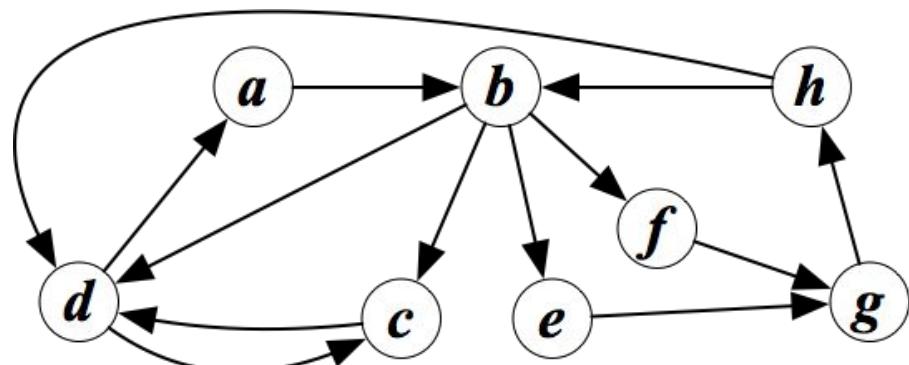
- **Back**: to an ancestor
- **Forward**: to a descendant
- **Cross**: to neither an ancestor nor descendant



Directed DFS: non-tree edges

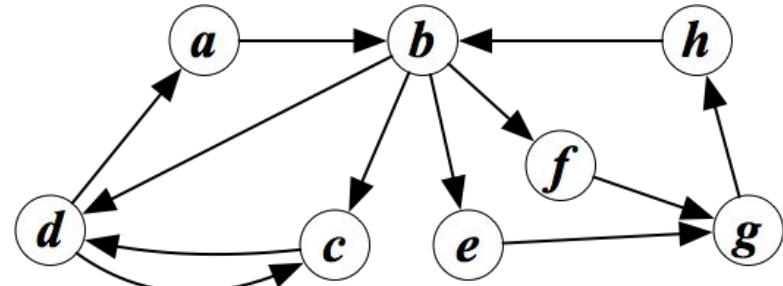
3 types:

- **Back**: to ancestor
- **Forward**: to descendant
- **Cross**: to neither ancestor nor descendant



Directed BFS: non-tree edges

- One level down
(tree or non-tree)
 - Why not > 1 ?
same reason as undirected
(would add to next level)
- Same level (non-tree)
- Any levels up (non-tree)



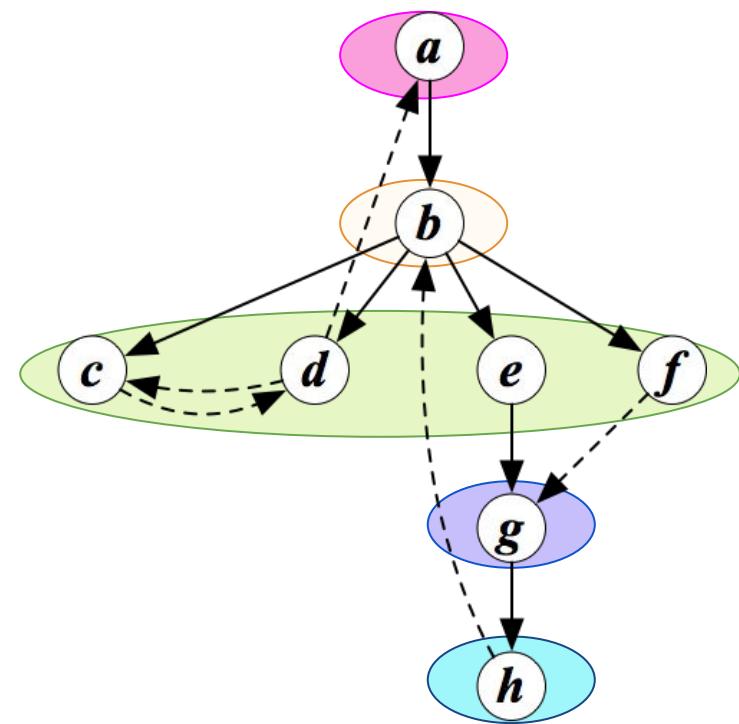
layer 0

layer 1

layer 2

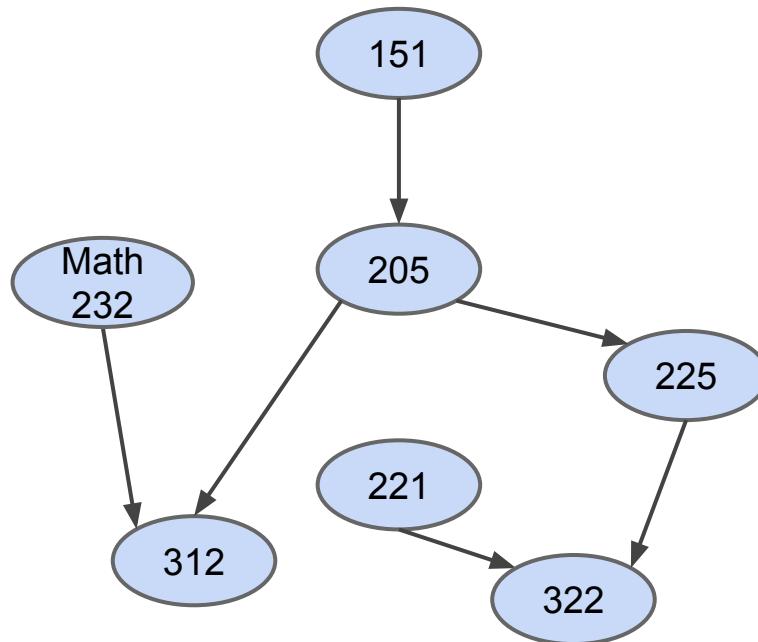
layer 3

layer 4



Directed acyclic graphs

- DAG: directed acyclic graph
 - No (directed) cycles
- Models dependencies

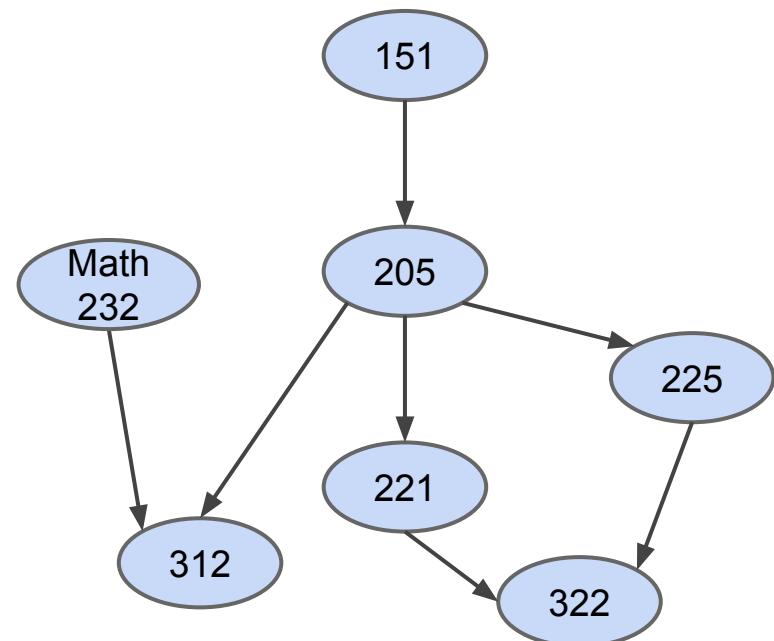


DAGs & topological sorting

- **topological ordering:** ordering of vertices such that all edges go “forward”

$$v_1, v_2, \dots, v_n$$

- If $(v_i, v_j) \in E$, then $i < j$
- Example: ordering of classes so prerequisites satisfied
- Question: topological ordering of DAG always possible?



DAGs & topological sorting

- Claim: topological sorting => DAG
- Proof (sketch):

Assume G has topological sort v_1, v_2, \dots, v_n such that

if $(v_i, v_j) \in E$, then $i < j$

Suppose, for contradiction, G has cycle $u_1, u_2, \dots, \dots, u_k$.

By definition of cycle/path, there exist edges for each pair u_a, u_{a+1} .

By definition of topological sort, only forward edges

=> u_i before u_j if $i < j$ in the sort

But, by definition of cycle, $u_1 = u_k$ giving the contradiction:

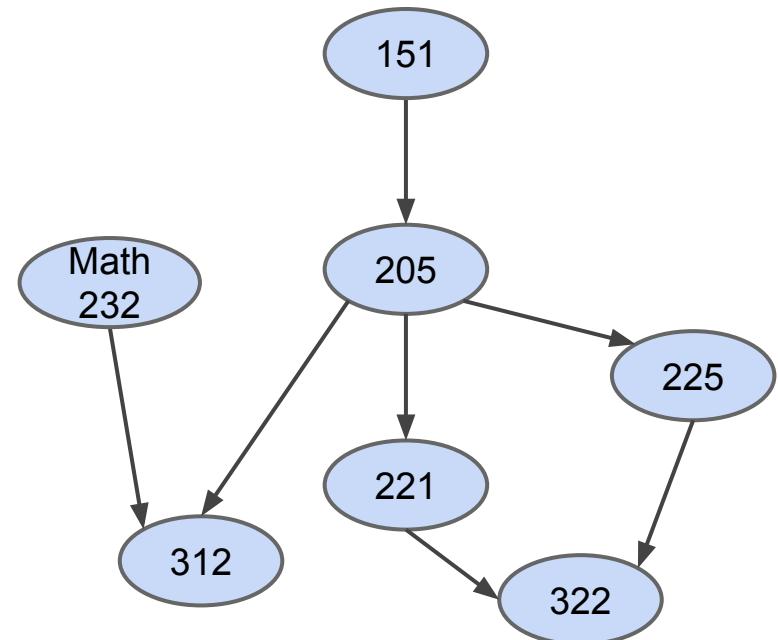
u_{k-1}, u_k forward edge, so u_{k-1} before u_k in the sort

$k = 1$, so u_{k-1} before u_1 in the sort

=> contradiction!

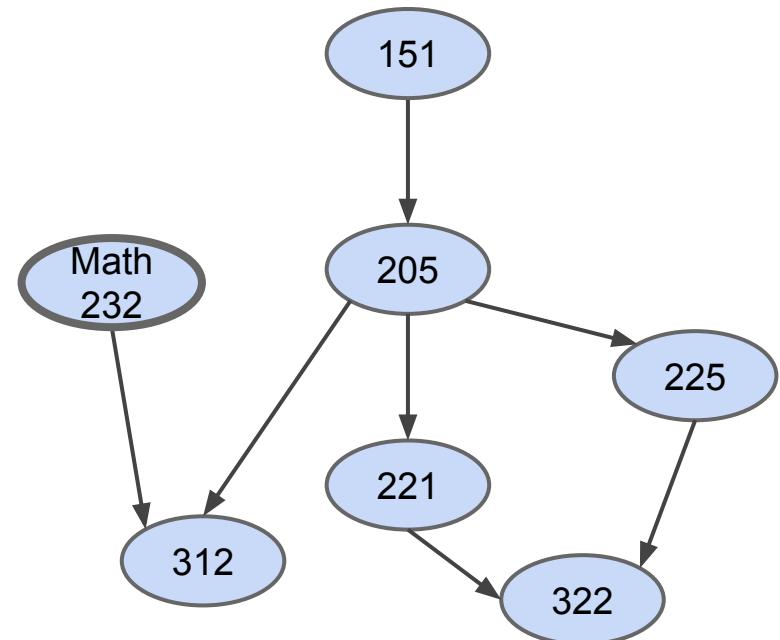
DAGs & topological sorting

- What about the other way?
- Does DAG => topological sorting?



DAGs & topological sorting

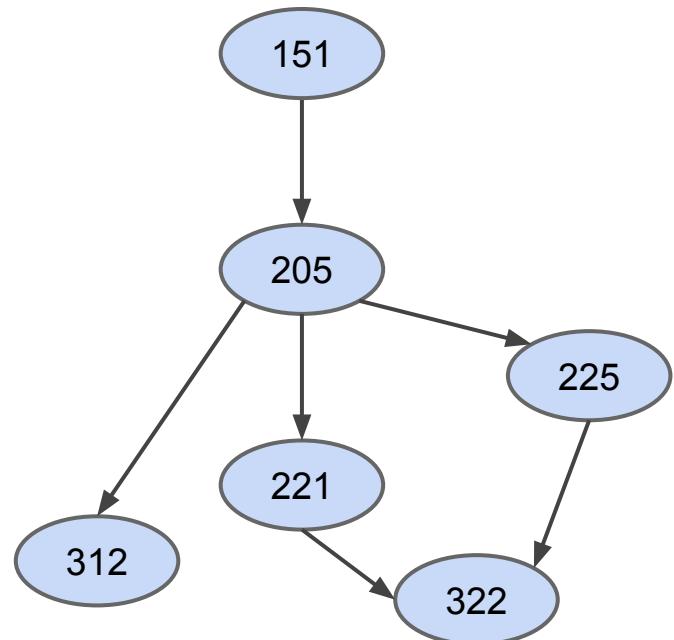
- What about the other way?
- Does DAG => topological sorting?



DAGs & topological sorting

- What about the other way?
- Does DAG => topological sorting?

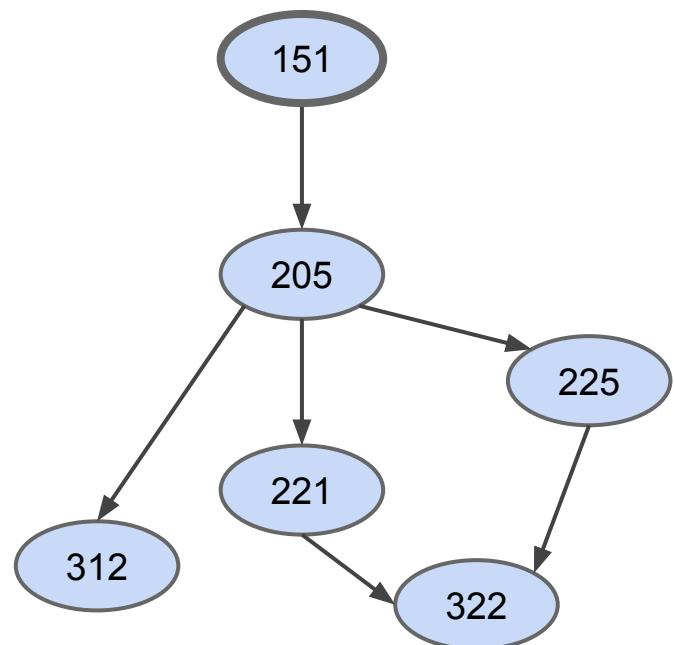
Math
232



DAGs & topological sorting

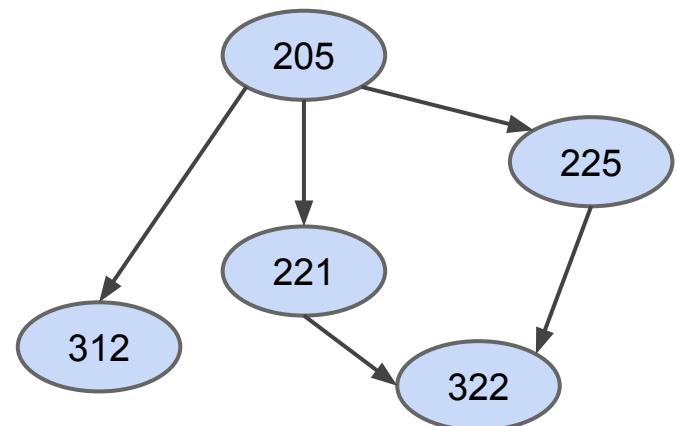
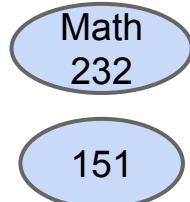
- What about the other way?
- Does DAG => topological sorting?

Math
232



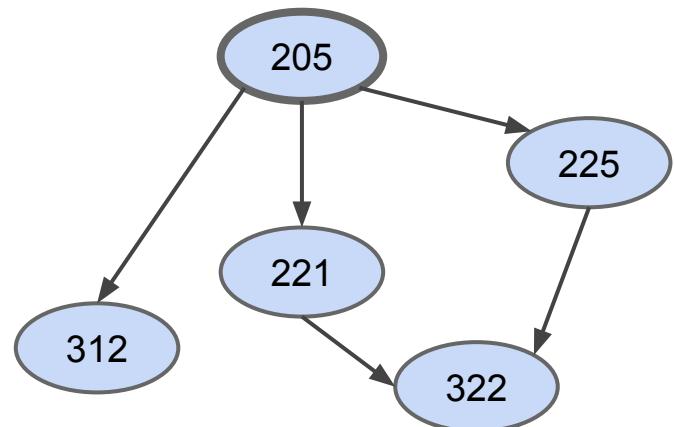
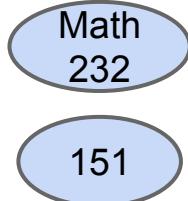
DAGs & topological sorting

- What about the other way?
- Does DAG => topological sorting?



DAGs & topological sorting

- What about the other way?
- Does DAG => topological sorting?



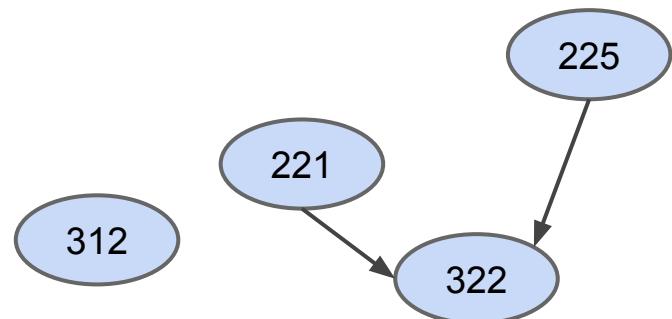
DAGs & topological sorting

- What about the other way?
- Does DAG => topological sorting?

Math
232

151

205



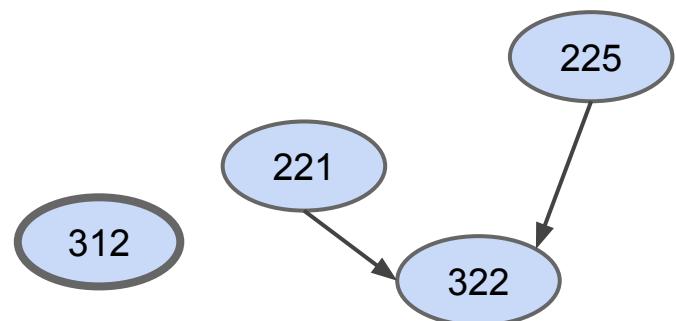
DAGs & topological sorting

- What about the other way?
- Does DAG => topological sorting?

Math
232

151

205



DAGs & topological sorting

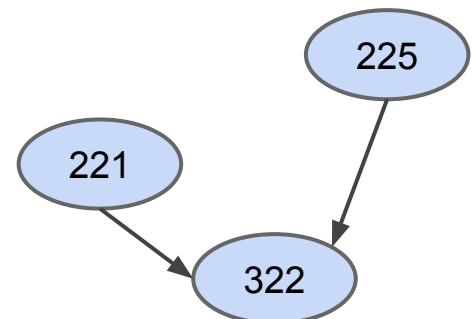
- What about the other way?
- Does DAG => topological sorting?

Math
232

151

205

312



DAGs & topological sorting

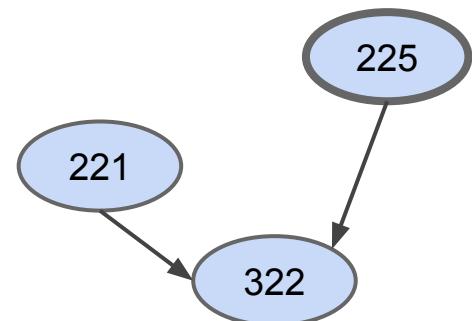
- What about the other way?
- Does DAG => topological sorting?

Math
232

151

205

312



DAGs & topological sorting

- What about the other way?
- Does DAG => topological sorting?



DAGs & topological sorting

- What about the other way?
- Does DAG => topological sorting?



DAGs & topological sorting

- What about the other way?
- Does DAG => topological sorting?



DAGs & topological sorting

- What about the other way?
- Does DAG => topological sorting?



DAG => topological ordering?

Claim: Every DAG has a vertex with no incoming edges.

Proof: (*feels familiar...*)

Suppose, for a contradiction, all v in DAG have incoming edges.

Construct a sequence of $n+1$ vertices $v_1, v_2, \dots, v_n, v_{n+1}$ such that v_{i+1} is an incoming neighbor to v_i .

There are $n+1$ variables (“nicknames”) for n vertices.

By the Pigeonhole Principle, there must be some vertex assigned to >1 variable (some vertex has multiple “nicknames”, some hole has multiple pigeons).

Thus, there is a cycle, giving the contradiction.

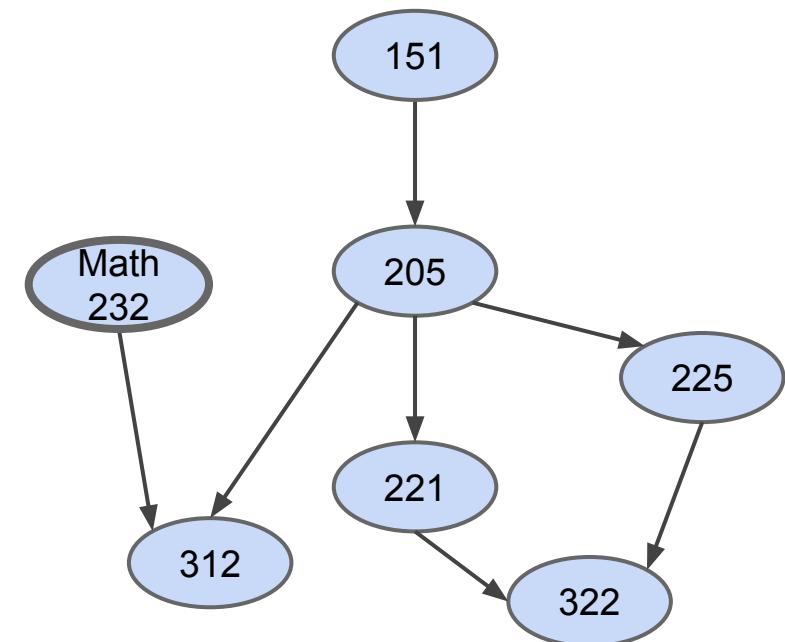


DAG <=> topological ordering

1. Every DAG has a vertex with no incoming edges.
2. Removing any vertex from DAG => still DAG.
3. Vertex with no incoming edges can be first in topological ordering.

... by induction ...

Theorem
 G is a DAG
if and only if
 G has a topological ordering.



Algorithm

topo-sort(DAG $G = (V, E)$):

init sequence $S = ()$

while $V \neq \emptyset$

 find $v \in V$ with no incoming edges

 place v at end of S

 remove v and its (outgoing) edges from V

return S

Correct?

Running time? $O(nm)$

Optimization for topological sorting

topo-sort(DAG $G = (V, E)$):

 init sequence $S = ()$

while $V \neq \emptyset$

 find $v \in V$ with no incoming edges

 place v at end of S

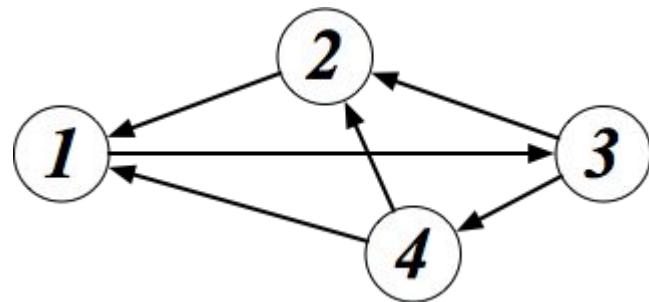
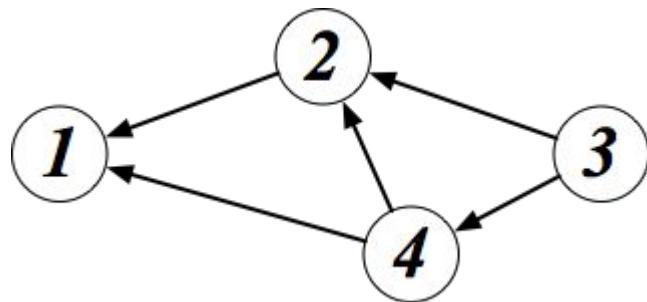
 remove v and its (outgoing) edges from V

return S

- don't search every time for nodes w/o incoming edges
- keep and update incoming edge count for each node:
 - Initialization: $O(m+n)$
 - Update: constant-time
- keep set of nodes with no incoming edges
 - add node when its count becomes zero
- Running time: $O(m+n)$

Directed graph: connectivity

- graph is ***strongly connected***:
exists path for every *ordered* pair of vertices



Directed graph: connectivity

- graph: ***strongly connected***:
exists path for all *ordered* vertex pairs
- ***strongly connected component (SCC)***:
maximal set of vertices with path for all *ordered* vertex pairs

