CS3000: Algorithms & Data
Paul Hand

Lecture 15:

- Depth First Search
- Topological Sorting
- Shortest Paths

Mar 13, 2019

Depth-First Search (DFS)

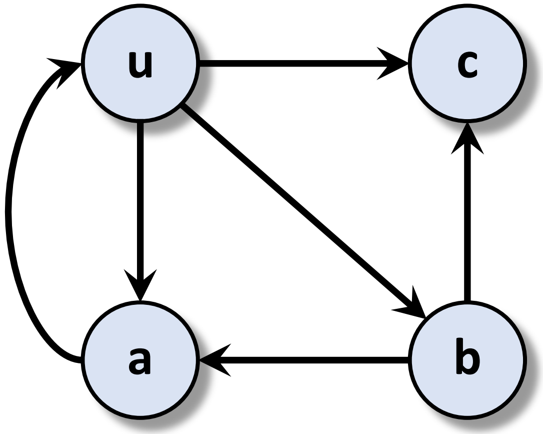
Exploring a Graph

- **Problem:** Is there a path from s to t?
- **Idea:** Explore all nodes reachable from s.

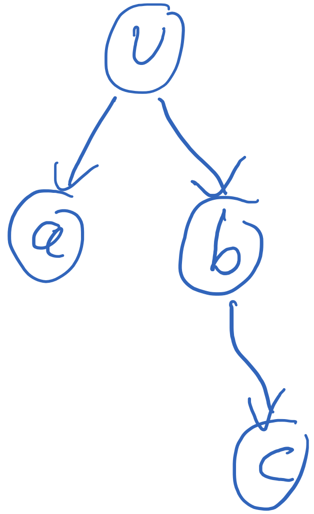
- Two different search techniques:
  - **Breadth-First Search:** explore nearby nodes before moving on to farther away nodes
  - **Depth-First Search:** follow a path until you get stuck, then go back

Depth-First Search

```
G = (V,E) is a graph
explored[u] = 0 ∀u

DFS(u):
  explored[u] = 1

  for ((u,v) in E):
    if (explored[v]=0):
      parent[v] = u
      DFS(v)
```



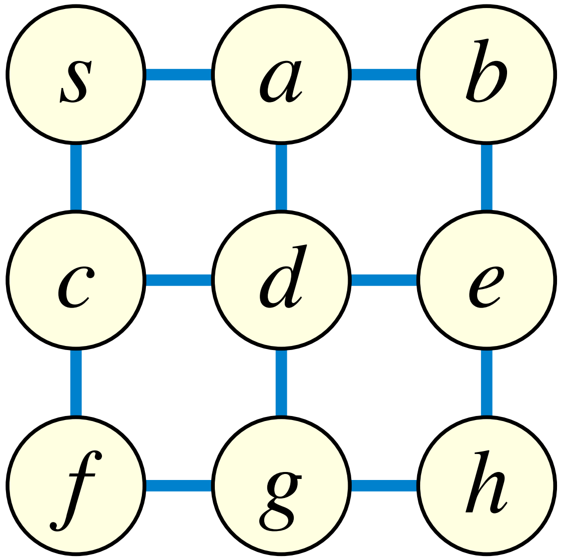
DFS Tree



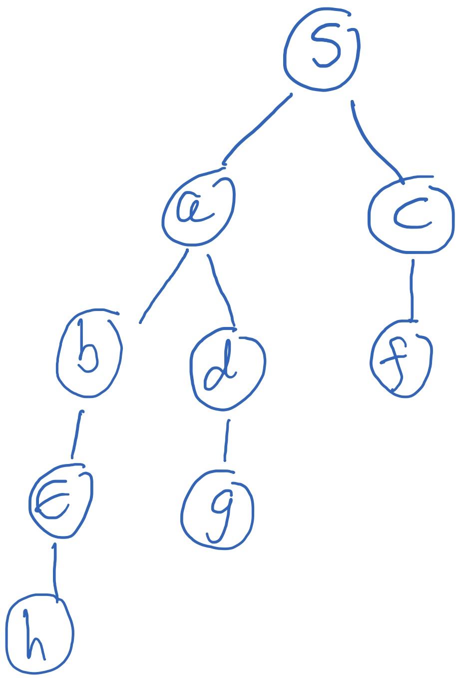
Choose edges by alphabetical order

Activity: Draw the BFS and DFS Trees (starting at s)

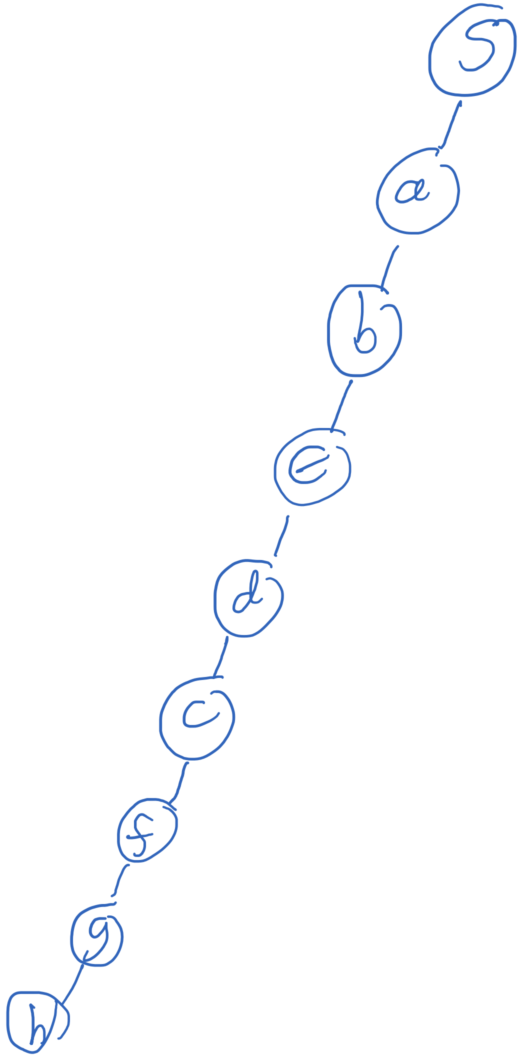
input



BFS tree

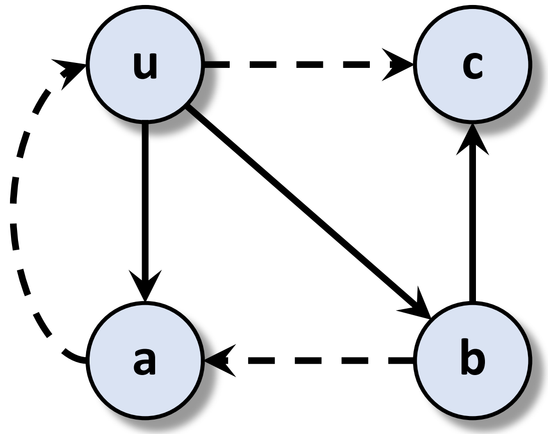


DFS Tree



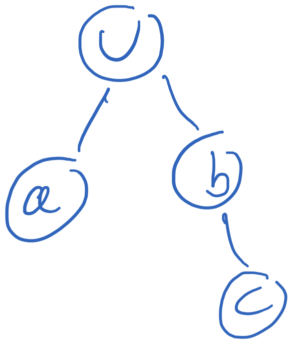
Depth-First Search

- **Fact:** The parent-child edges form a (directed) tree
- **Each edge has a type:**
  - **Tree edges:** $(u, a), (u, c), (c, b)$
    - These are the edges that explore new nodes
  - **Forward edges:** (u, b)
    - Ancestor to descendant
  - **Backward edges:** (a, u)
    - Descendant to ancestor
  - **Cross edges:** (c, a)
    - No ancestral relation



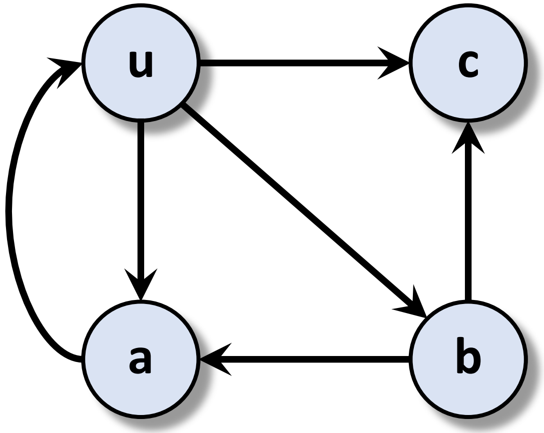
Pre-Ordering

- Order the vertices by when they were **first** visited by DFS



U, a, b, C

```
G = (V,E) is a graph
explored[u] = 0 ∀u

DFS(u):
  explored[u] = 1

  pre-visit(u)

  for ((u,v) in E):
    if (explored[v]=0):
      parent[v] = u
      DFS(v)
```

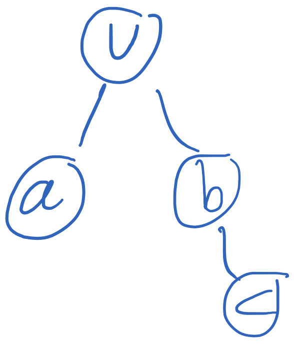


Vertex	Pre-Order
u	1
a	2
b	3
c	4

- Maintain a counter `clock`, initially set `clock = 1`
- `pre-visit(u):`
  `set preorder[u]=clock, clock=clock+1`

Post-Ordering

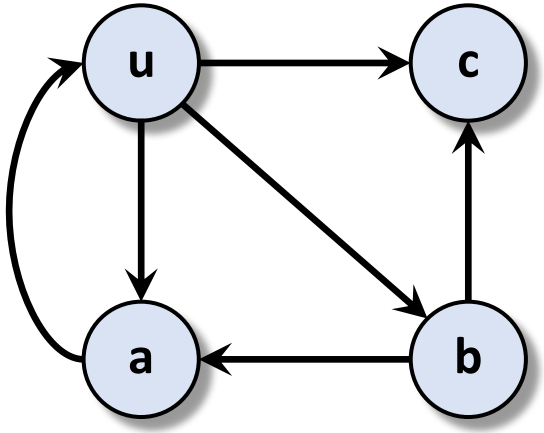
- Order the vertices by when they were **last** visited by DFS



Post order?

a, c b, u

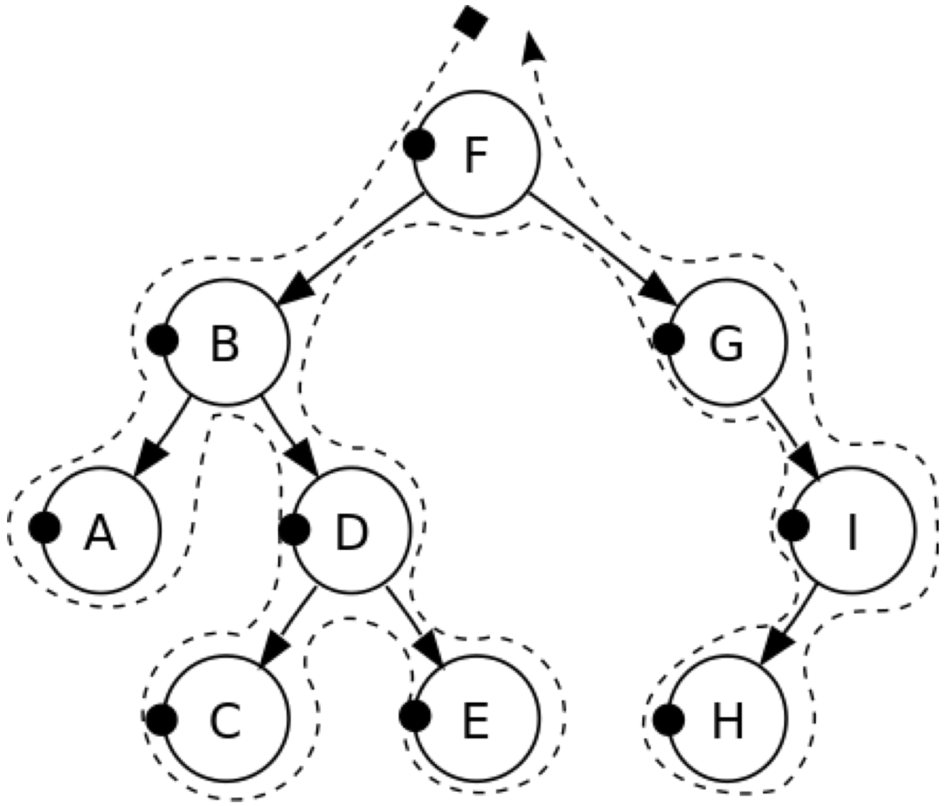
```
G = (V,E) is a graph
explored[u] = 0 ∀u

DFS(u):
  explored[u] = 1

  for ((u,v) in E):
    if (explored[v]=0):
      parent[v] = u
      DFS(v)

  post-visit(u)
```



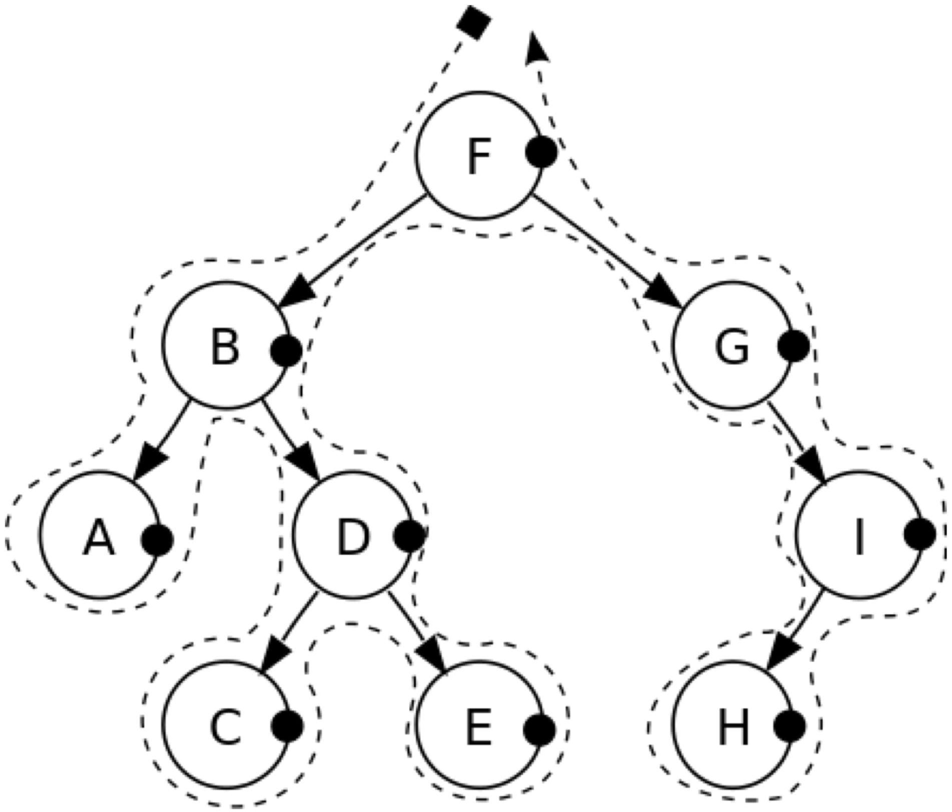
Vertex	Post-Order
u	4
a	1
b	3
c	2

- Maintain a counter **clock**, initially set `clock = 1`
- **post-visit(u):**
  `set postorder[u]=clock, clock=clock+1`

Preorder versus postorder



Pre-order: F, B, A, D, C, E, G, I, H.

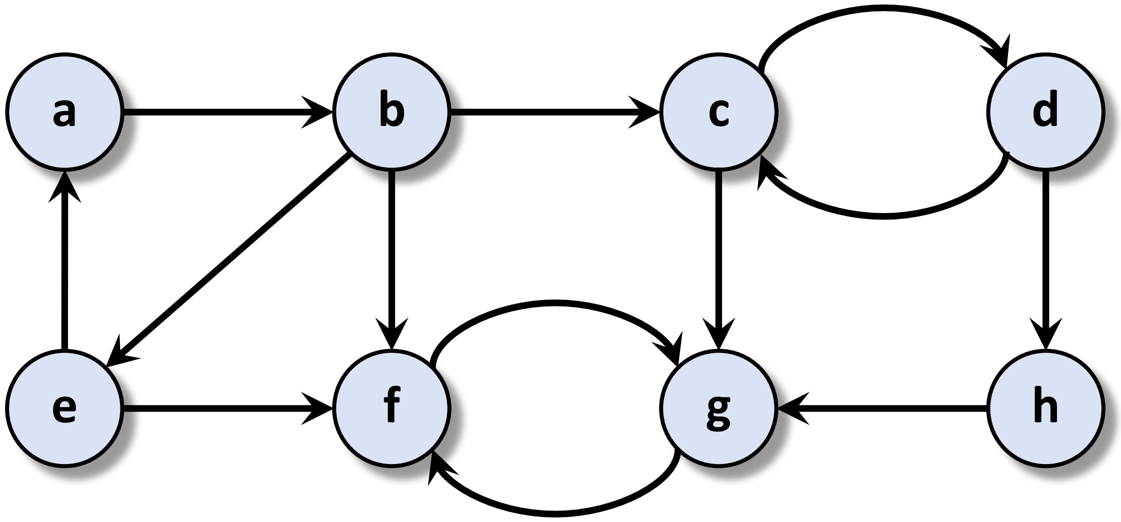


Post-order: A, C, E, D, B, H, I, G, F.

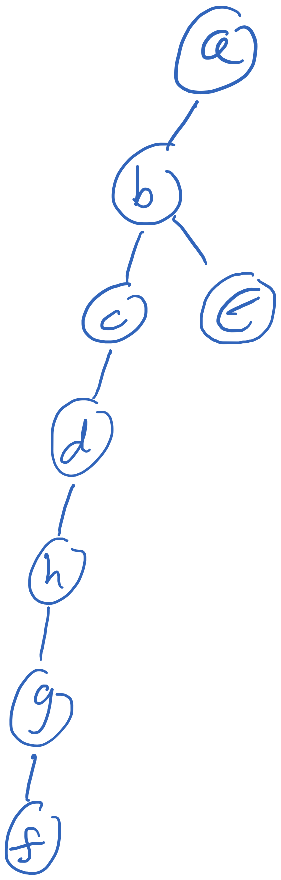
Activity

① Form DFS Tree
② Read off post order

- Compute the **post-order** of this graph
    - DFS from $\boldsymbol{a}$, search in alphabetical order



DFS Tree



Post order

f g h d c e b a

1 2 3 4 5 6 7 8

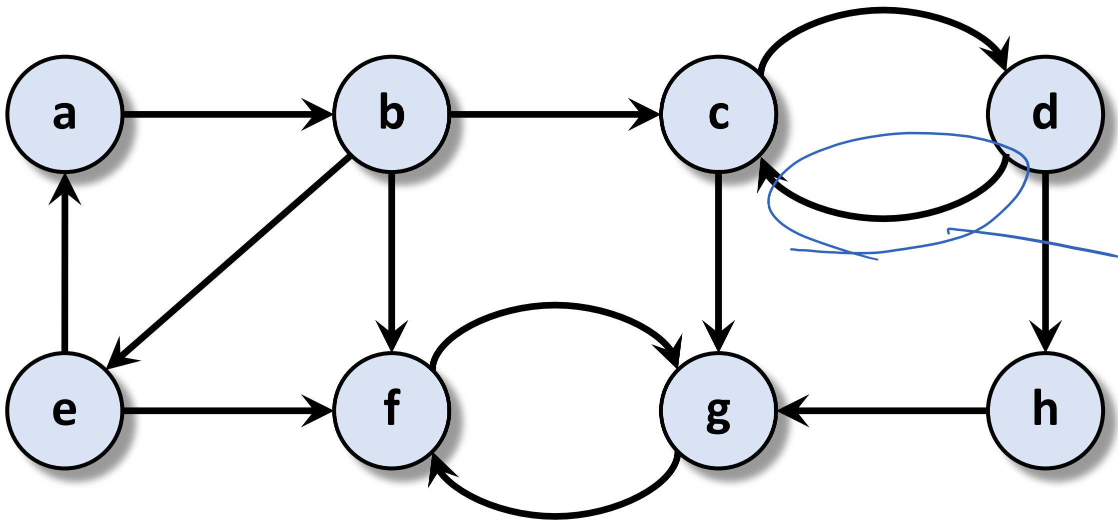
Vertex	a	b	c	d	e	f	g	h
Post-Order	8	7	5	4	6	1	2	3

Activity

Edge (u,v) from u to v

finished processing u before finished processing v

- **Observation:** if postorder[u] < postorder[v] then (u,v) is a backward edge



backward

(d, c)

postorder of d < postorder of c

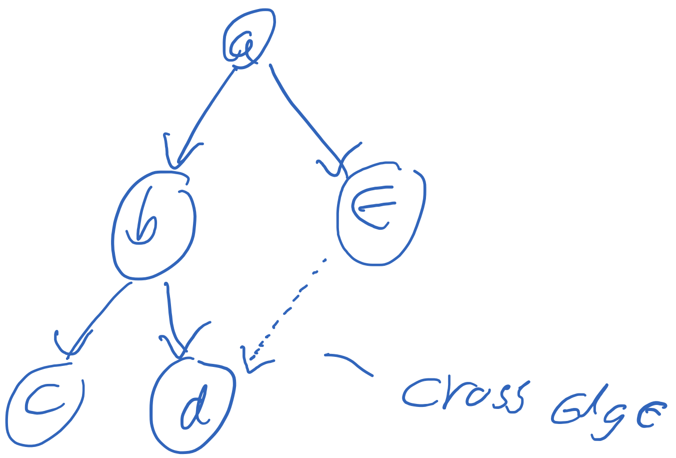
backward edge

Vertex	a	b	c	d	e	f	g	h
Post-Order	8	7	5	4	6	1	2	3

Observation about postordering

- **Observation:** if postorder[u] < postorder[v] then (u,v) is a backward edge
  - DFS(u) can't finish until its children are finished
    - If (u,v) is a tree edge, then postorder[u] > postorder[v]
    - If (u,v) is a forward edge, then postorder[u] > postorder[v]
  - If postorder[u] < postorder[v], then DFS(u) finishes before DFS(v), thus DFS(v) is not called by DFS(u)
  - When we ran DFS(u), we must have had explored[v]=1
    - Thus, DFS(v) started before DFS(u)
  - DFS(v) started before DFS(u) but finished after
    - Can only happen for a backward edge

Example



(e,d) is a cross edge

If post(e) < post(d)

e started before d

but

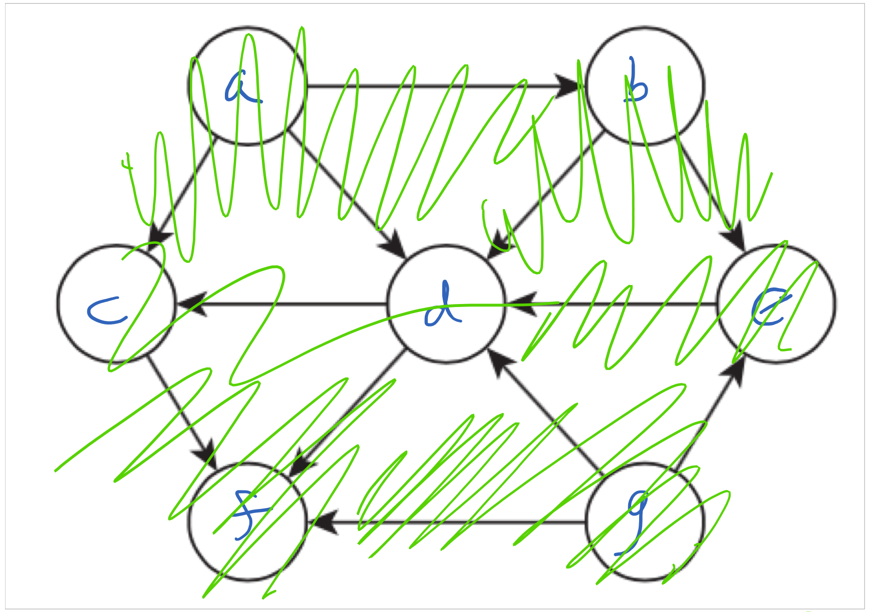
finished after

Fast Topological Ordering

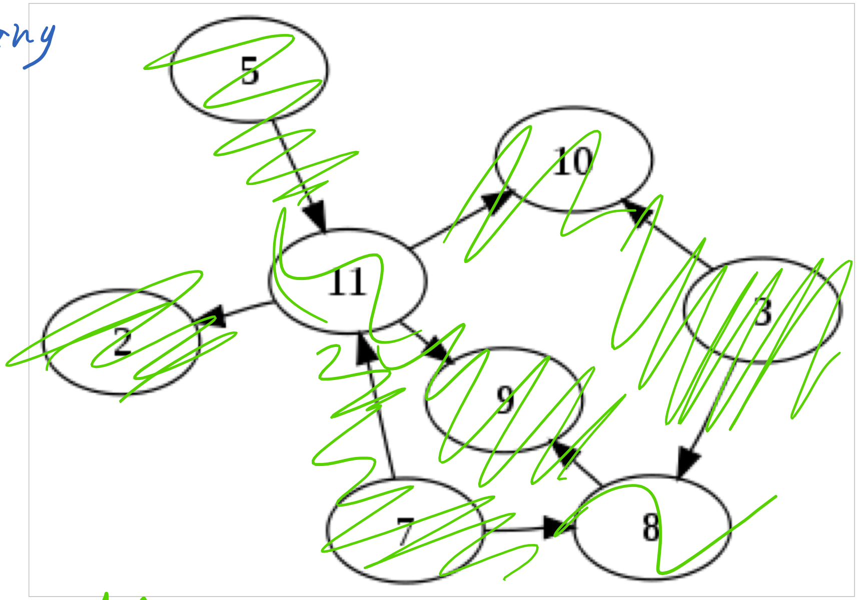
Topological Ordering (TO)

- **DAG:** A directed graph with no directed cycles.
- Are these DAGs?

Search for cycles
Remove vertices
that could not
be part of any
cycle.



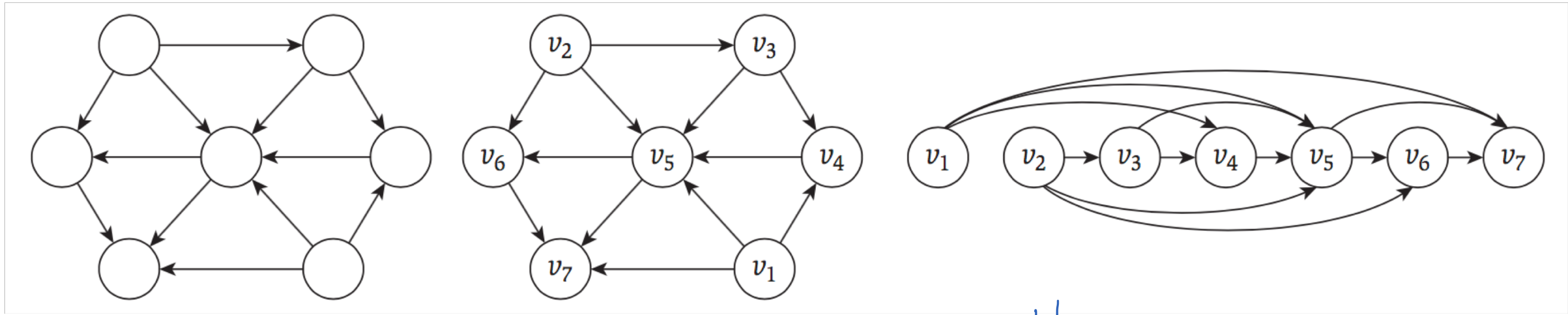
No cycles DAG



No cycles DAG

Topological Ordering (TO)

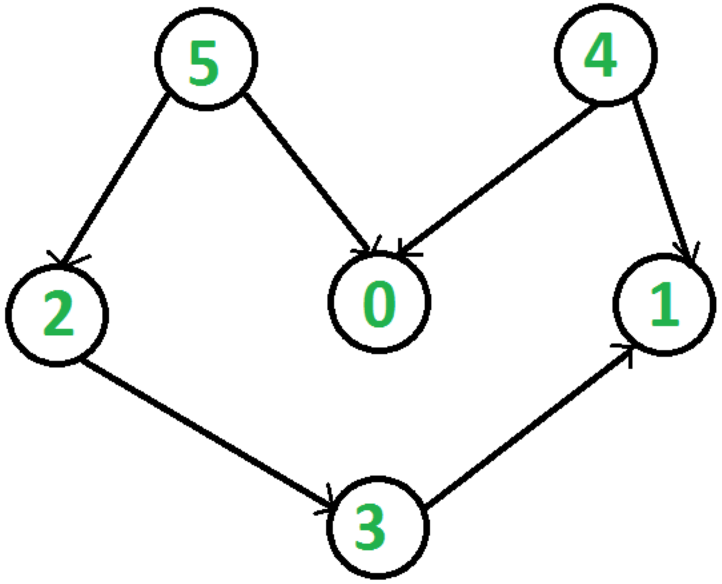
- **DAG:** A directed graph with no directed cycles
- Any DAG can be **toplogically ordered**
  - Label nodes $v_1, \ldots, v_n$ so that $(v_i, v_j) \in E \Longrightarrow j > i$



arrange nodes so that they all are in a line
and all edges go to the right

Activity

- Come up with two different topologically orderings of the following graph

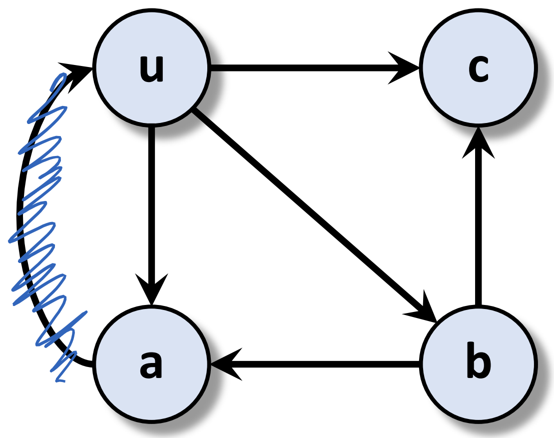


5 2 3 4 1 0

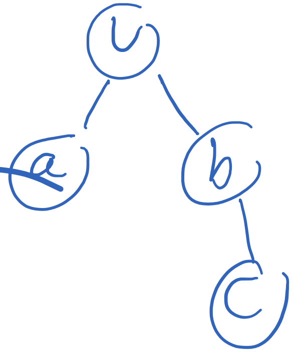
5 2 3 4 1 0

4 5 2 3 1 0

Algorithm for Topological Ordering

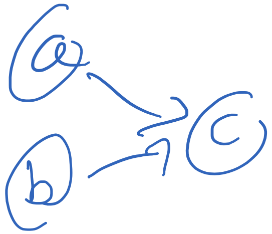


- **Claim:** ordering nodes by decreasing postorder gives a topological ordering
- **Proof:**
  - A DAG has no backward edges (Such an edge would form a cycle)
  - Suppose this is **not** a topological ordering
    - That means there exists an edge (u,v) such that postorder[u] < postorder[v]
    - We showed that any such (u,v) is a backward edge
    - But there are no backward edges, contradiction!



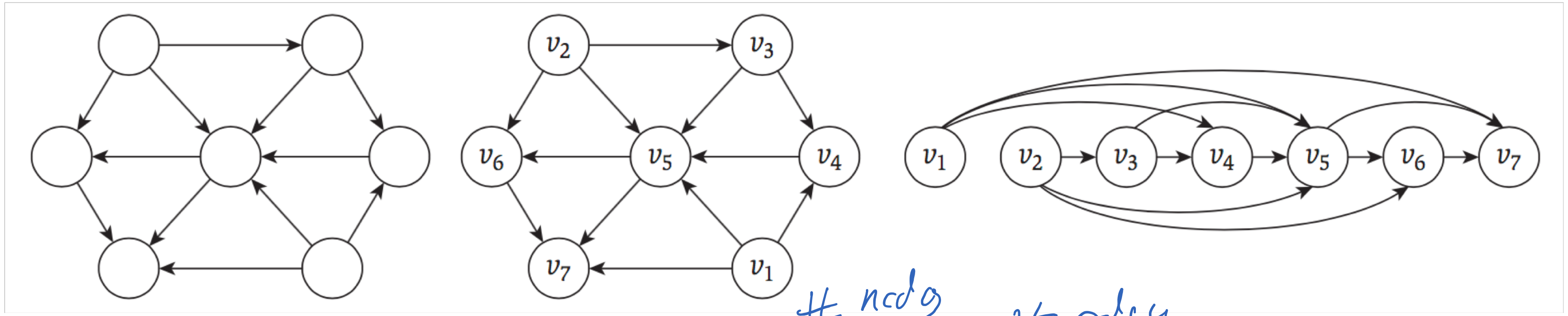
Post order
a, c, b, u

Reversed Postorder
u, b, c a



Topological Ordering (TO)

- **DAG:** A directed graph with no directed cycles
- Any DAG can be **toplogically ordered**
  - Label nodes $v_1, \dots, v_n$ so that $(v_i, v_j) \in E \Longrightarrow j > i$



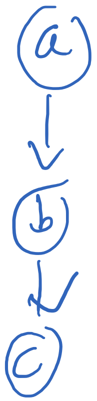
\# nodes \# edges

- Can compute a TO in $O(n + m)$ time using DFS
  - Reverse of post-order is a topological order

Activity

- Come up with a DAG with 3 nodes such that the preordering is not a topological ordering.

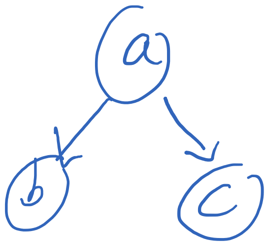
Does this work?



Preordering

a, b, c

What about?

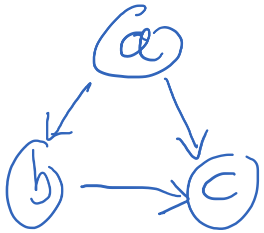


Preorder

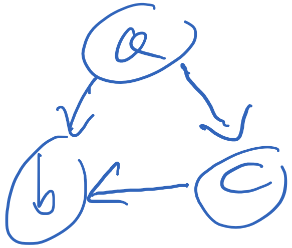
a, b, c

Is topo. order

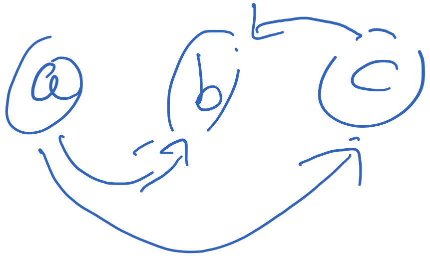
What about?

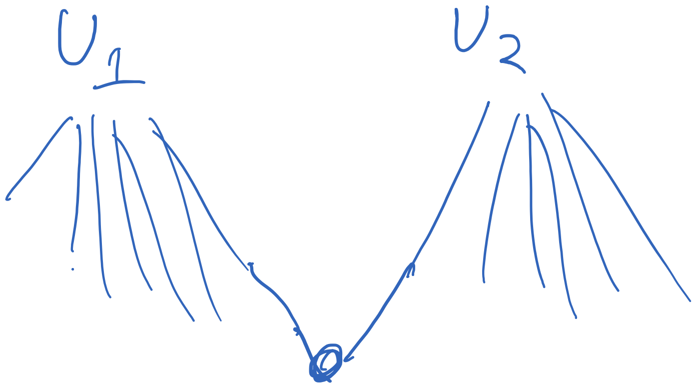


a, b, c



a, b, c





Consider FB Social graph.

Given user 1 & user 2

You want to determine if there is a path between user 1 & user 2

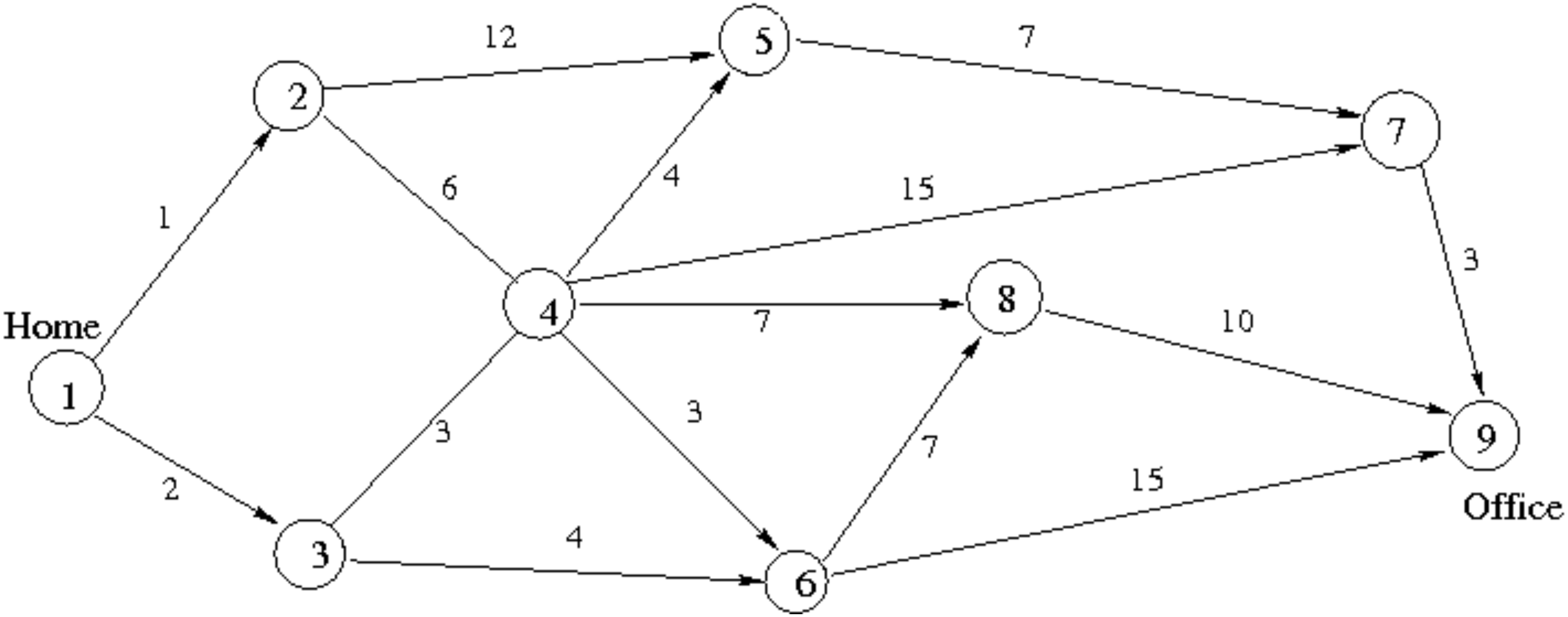
Would you use BFS or DFS to do this?

~~Shortest Paths~~

DFS will make this query as long as worst case

Activity: Find the shortest path

19



Weighted Graphs

- **Definition:** A weighted graph $G = (V, E, \{w(e)\})$
  - V is the set of vertices
  - $E \subseteq V \times V$ is the set of edges
  - $w_e \in \mathbb{R}$ are edge weights/lengths/capacities
  - Can be directed or undirected

- **Today:**
  - Directed graphs (one-way streets)
  - Strongly connected (there is always some path)
  - Non-negative edge lengths ($\ell(e) \geq 0$)

Shortest Paths

- The length of a path $P = v_1 - v_2 - \cdots - v_k$ is the sum of the edge lengths

- The distance $d(s,t)$ is the length of the shortest path from s to t
- **Shortest Path:** given nodes $s,t \in V$, find the shortest path from s to t
- **Single-Source Shortest Paths:** given a node $s \in V$, find the shortest paths from s to **every** $t \in V$

Structure of Shortest Paths

- If $(u, v) \in E$, then $d(s, v) \leq d(s, u) + \ell(u, v)$ for every node $s \in V$

- If $(u, v) \in E$, and $d(s, v) = d(s, u) + \ell(u, v)$ then there is a shortest $s \rightsquigarrow v$-path ending with (u, v)