CS3000: Algorithms & Data
Jonathan Ullman

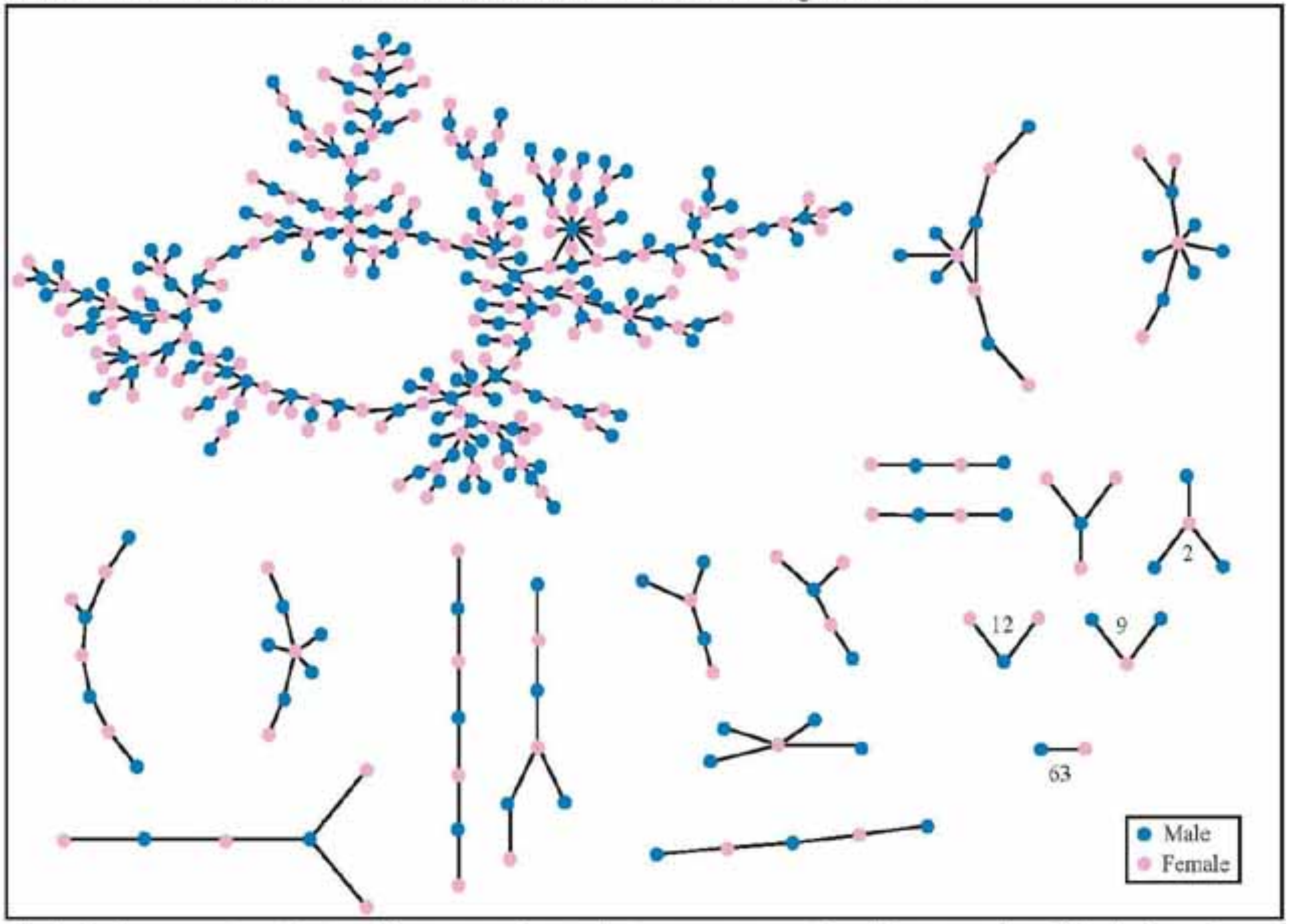
Lecture 9:

- Graphs
- Graph Traversals: DFS
- Topological Sort

Feb 5, 2020

What’s Next

The Structure of Romantic and Sexual Relations at "Jefferson High School"



Each circle represents a student and lines connecting students represent romantic relations occuring within the 6 months preceding the interview. Numbers under the figure count the number of times that pattern was observed (i.e. we found 63 pairs unconnected to anyone else).

What's Next

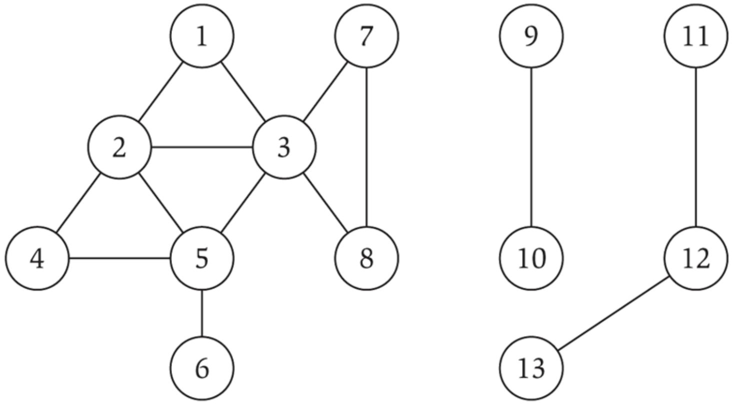
- **Graph Algorithms:**
  - **Graphs:** Key Definitions, Properties, Representations
  - **Exploring Graphs:** Breadth/Depth First Search
    - Applications: Connectivity, Bipartiteness, Topological Sorting
  - **Shortest Paths:**
    - Dijkstra
    - Bellman-Ford (Dynamic Programming)
  - **Minimum Spanning Trees:**
    - Borůvka, Prim, Kruskal
  - **Network Flow:**
    - Algorithms
    - Reductions to Network Flow

Graphs

Graphs: Key Definitions

- **Definition:** A directed graph $G = (V, E)$
  - V is the set of nodes/vertices
  - $E \subseteq V \times V$ is the set of edges
  - An edge is an ordered $e = (u, v)$ "from u to v"
- **Definition:** An undirected graph $G = (V, E)$
  - Edges are unordered $e = (u, v)$ "between u and v"

- **Simple Graph:**
  - No duplicate edges
  - No self-loops $e = (u, u)$



Ask the Audience

- How many edges can there be in a **simple** directed/undirected graph?

Graphs Are Everywhere

- Transportation networks
- Communication networks
- WWW
- Biological networks
- Citation networks
- Social networks
- …

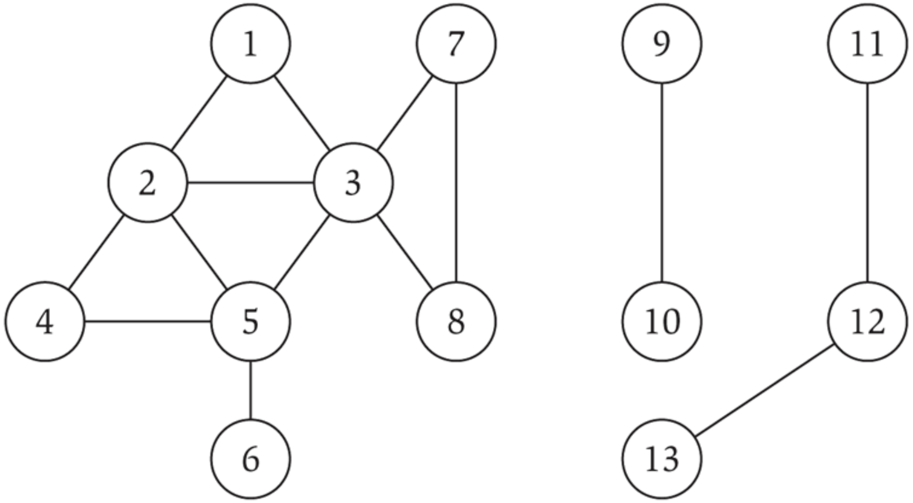
Paths/Connectivity

- A path is a sequence of consecutive edges in E
  - $P = \{(u, w_1), (w_1, w_2), (w_2, w_3), \ldots, (w_{k-1}, v)\}$
  - $P = u - w_1 - w_2 - w_3 - \cdots - w_{k-1} - v$
  - The length of the path is the # of edges

- An undirected graph is connected if for every two vertices $u, v \in V$, there is a path from u to v
- A directed graph is strongly connected if for every two vertices $u, v \in V$, there are paths from u to v and from v to u

Cycles

- A cycle is a path $v_1 - v_2 - \cdots - v_k - v_1$ and $v_1, \ldots, v_k$ are distinct



Ask the Audience

- Suppose an undirected graph G is connected
  - True/False? G has at least $n - 1$ edges

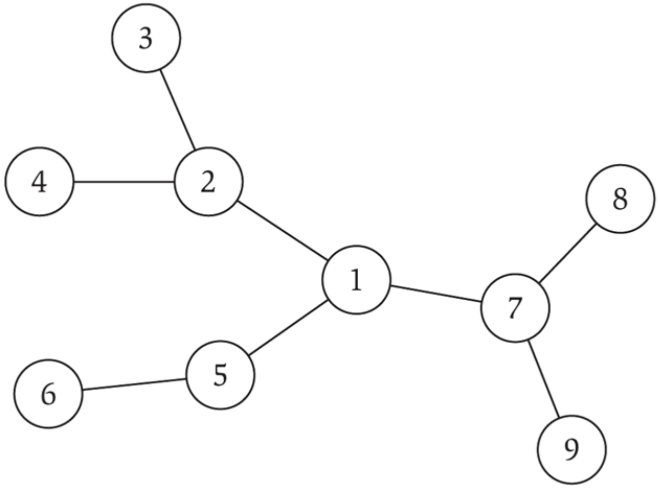
Ask the Audience

- Suppose an undirected graph G has $n-1$ edges
  - True/False? G is connected

Trees

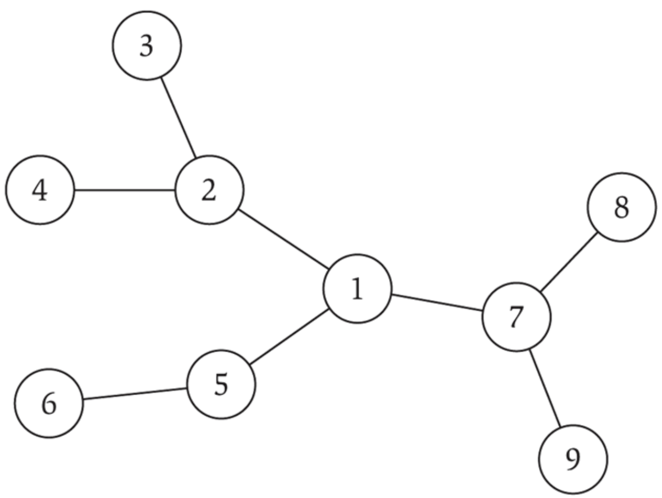
- A simple undirected graph G is a tree if:
  - G is connected
  - G contains no cycles

- **Theorem:** any two of the following implies the third
  - G is connected
  - G contains no cycles
  - G has $= n - 1$ edges

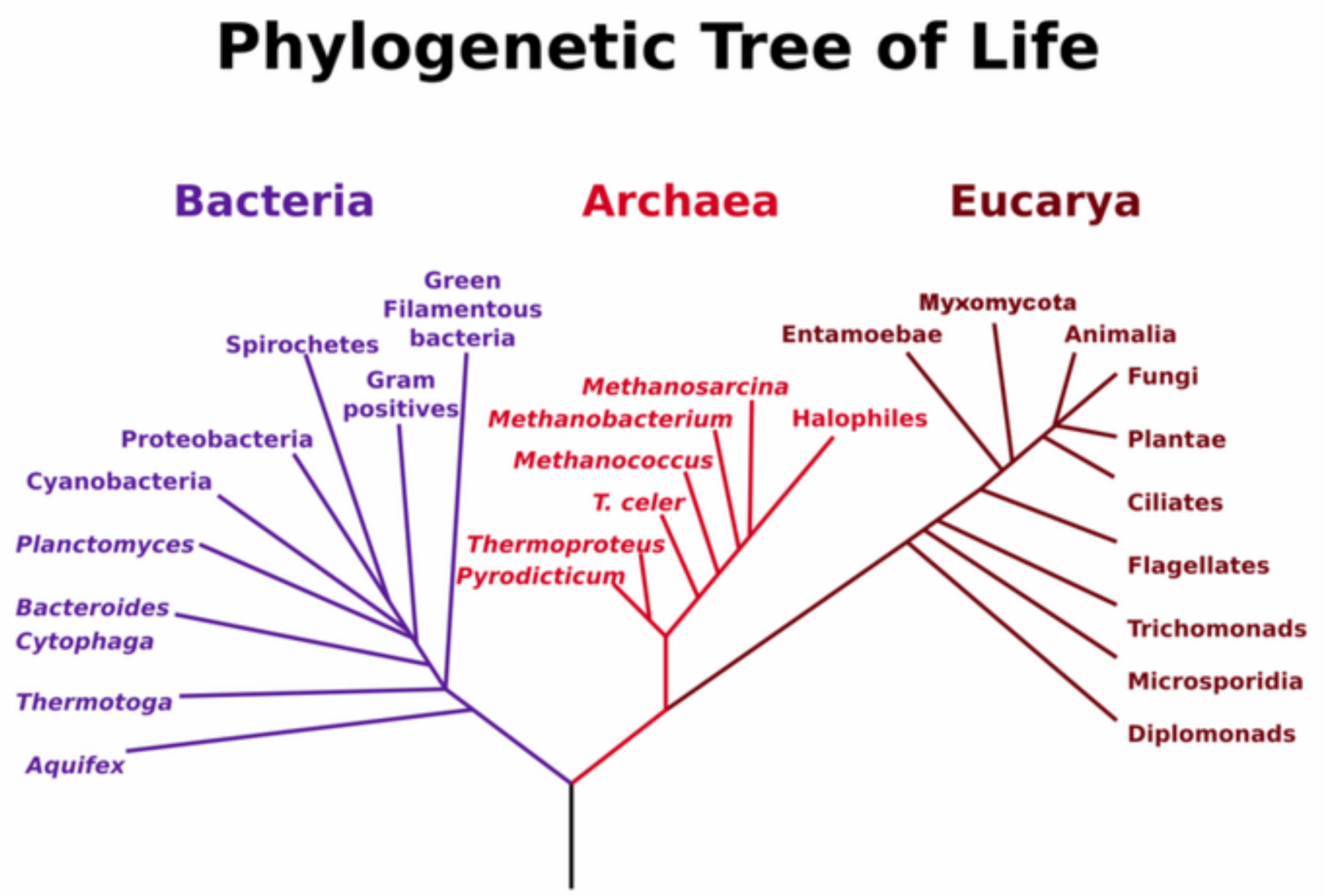


Trees

- Rooted tree: choose a root node r and orient edges away from r
  - Models **hierarchical structure**

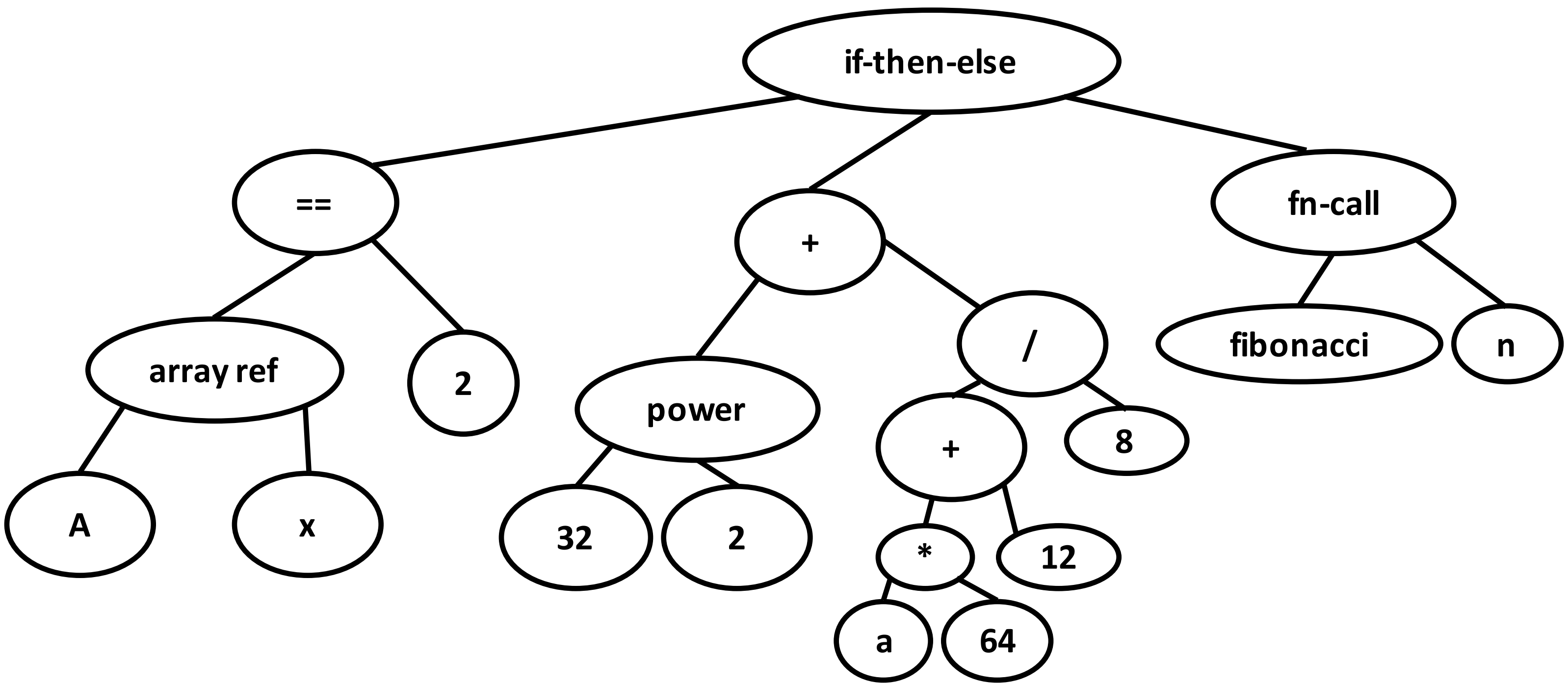


Phylogeny Trees



Parse Trees

```
if (A[x]==2) then
   (32² + (a*64 +12)/8)
else
   fibonacci(n)
```



Representing a Graph

Adjacency Matrices

- The adjacency matrix of a graph $G = (V, E)$ with n nodes is the matrix $A[1{:}n\,, 1{:}n]$ where

$$A[i,j] = \begin{cases} 1 & (i,j) \in E \\ 0 & (i,j) \notin E \end{cases}$$

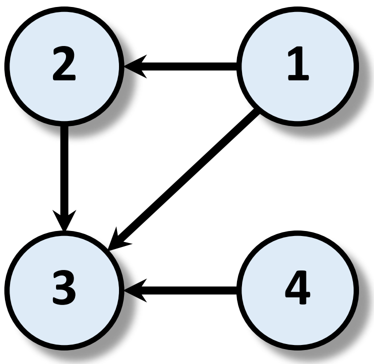
A	1	2	3	4
1	0	1	1	0
2	0	0	1	0
3	0	0	0	0
4	0	0	1	0

<u>Cost</u>

Space: $\Theta(V^2)$

Lookup: $\Theta(1)$ time

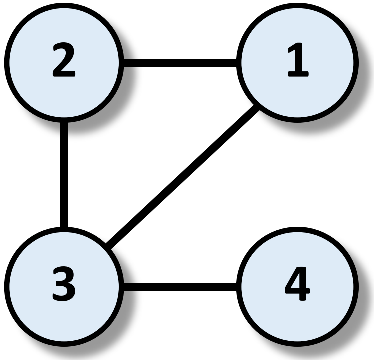
List Neighbors: $\Theta(V)$ time



Adjacency Lists (Undirected)

- The adjacency list of a vertex $v \in V$ is the list $A[v]$ of all u s.t. $(v, u) \in E$

$$A[1] = \{2,3\}$$
$$A[2] = \{1,3\}$$
$$A[3] = \{1,2,4\}$$
$$A[4] = \{3\}$$



Adjacency Lists (Directed)

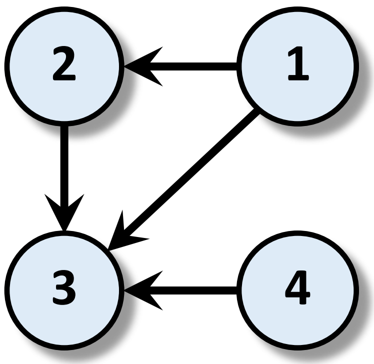
- The adjacency list of a vertex $v \in V$ are the lists
  - $A_{out}[v]$ of all u s.t. $(v, u) \in E$
  - $A_{in}[v]$ of all u s.t. $(u, v) \in E$

$A_{out}[1] = \{2,3\}$ $A_{in}[1] = \{\,\}$

$A_{out}[2] = \{3\}$ $A_{in}[2] = \{1\}$

$A_{out}[3] = \{\,\}$ $A_{in}[3] = \{1,2,4\}$

$A_{out}[4] = \{3\}$ $A_{in}[4] = \{\,\}$



Exploring a Graph

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

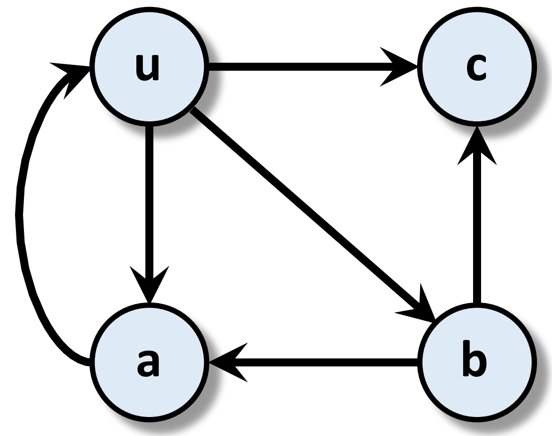
Exploring a Graph

- **BFS/DFS** are general templates for graph algorithms
  - Extensions of **Breadth-First Search**:
    - 2-Coloring (Bipartiteness)
    - Shortest Paths
    - Minimum Spanning Tree (Prim’s Algorithm)
  - Extensions of **Depth-First Search**:
    - Topological Sorting

Depth-First Search (DFS)

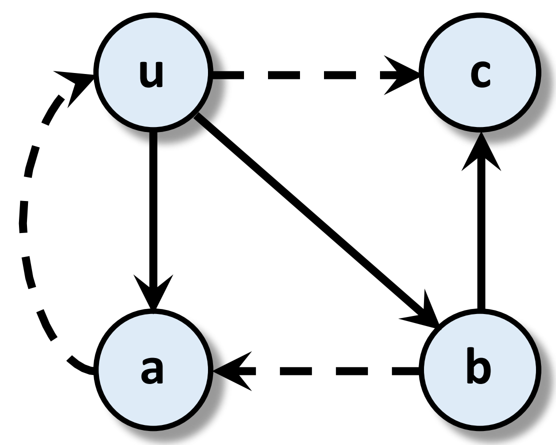
Depth-First Search

```
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)
```

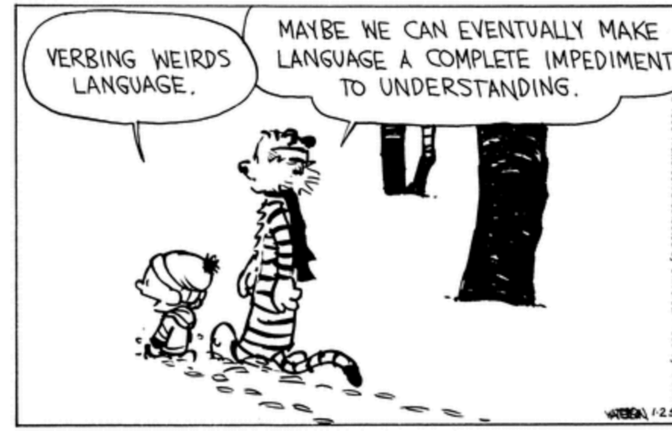


Depth-First Search

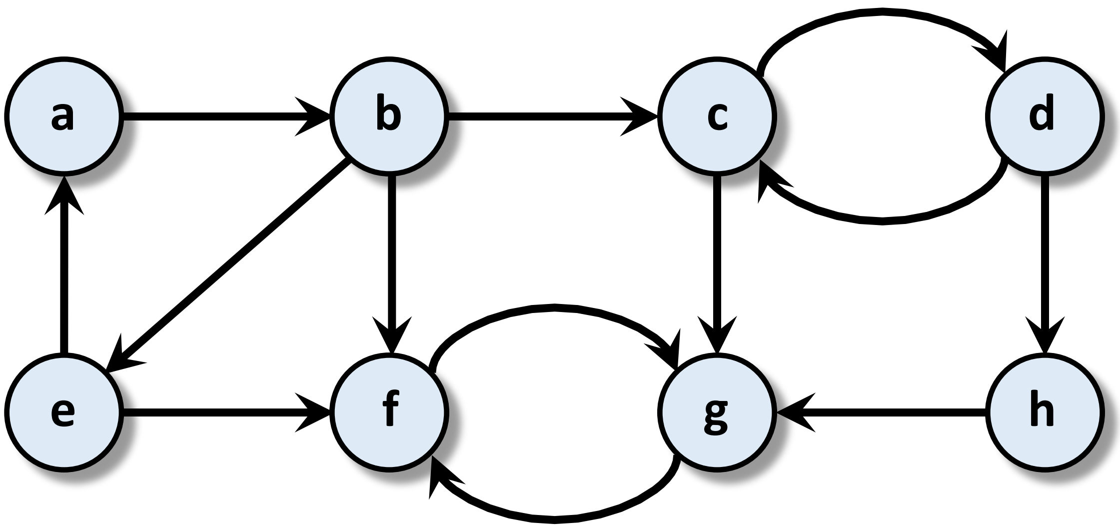
- **Fact:** The parent-child edges form a (directed) tree
- **Each edge has a type:**
  - **Tree edges:** $(u, a), (u, b), (b, c)$
    - These are the edges that explore new nodes
  - **Forward edges:** (u, c)
    - Ancestor to descendant
  - **Backward edges:** (a, u)
    - Descendant to ancestor
 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



Ask the Audience

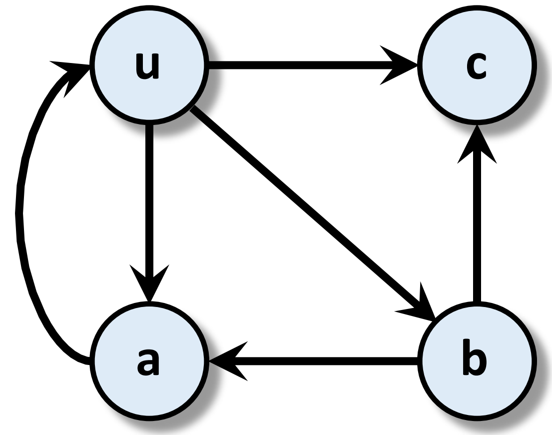


- DFS starting from node a
  - Search in alphabetical order
  - Label edges with {tree,forward,backward,cross}



Post-Ordering

```
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)
```

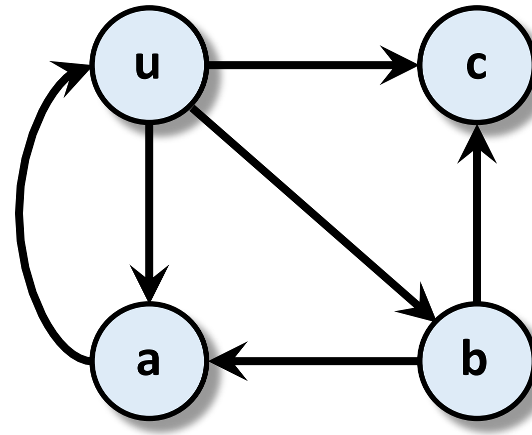


Vertex	Post-Order

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

Pre-Ordering

```
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)
```

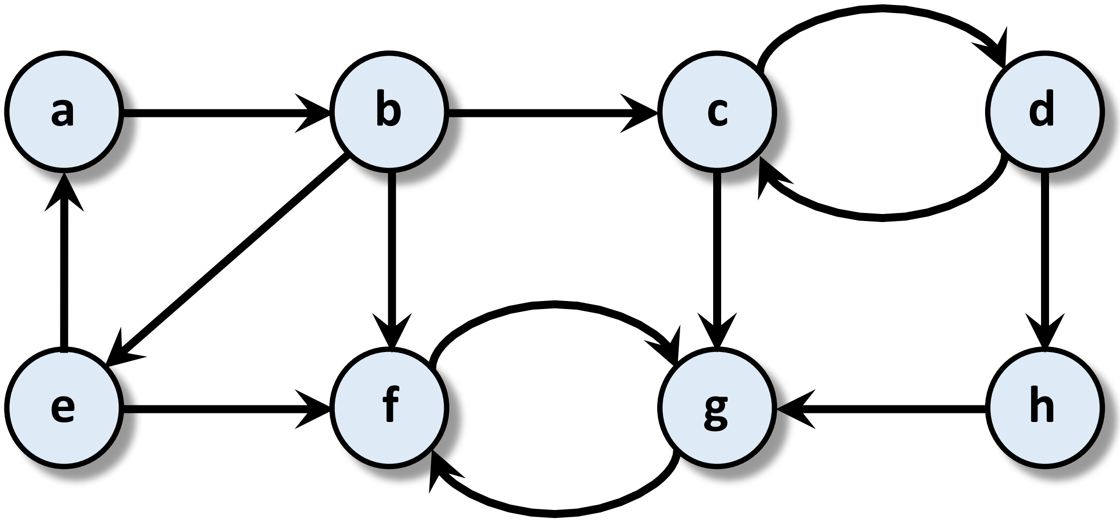


Vertex	Pre-Order

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

Ask the Audience

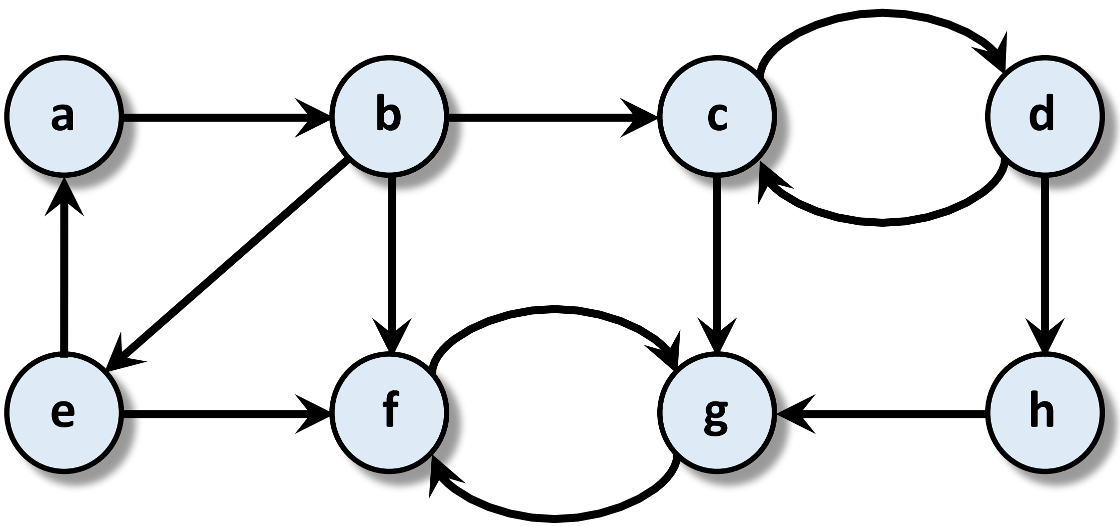
- Compute the **post-order** of this graph
  - DFS from a, search in alphabetical order



Vertex	a	b	c	d	e	f	g	h
Post-Order								

Ask the Audience

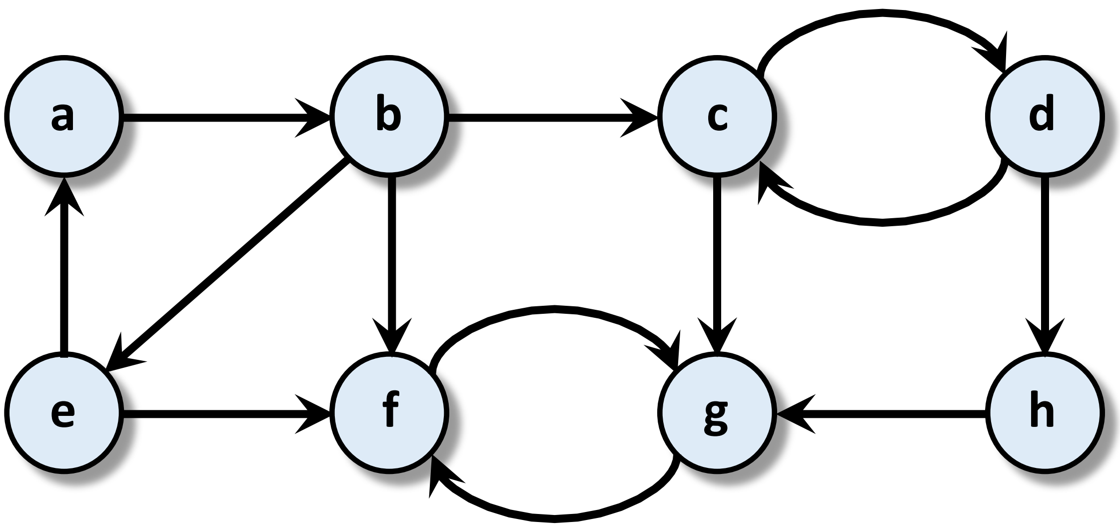
- Compute the **post-order** of this graph
  - DFS from a, search in alphabetical order



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

Ask the Audience

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



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

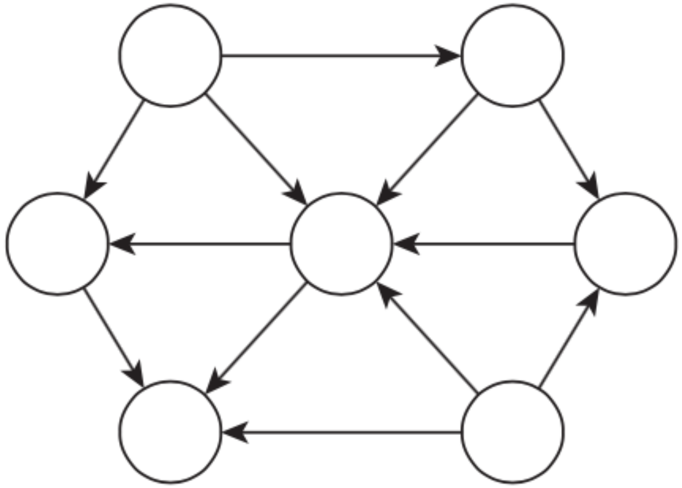
Ask the Audience

- **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

Topological Ordering

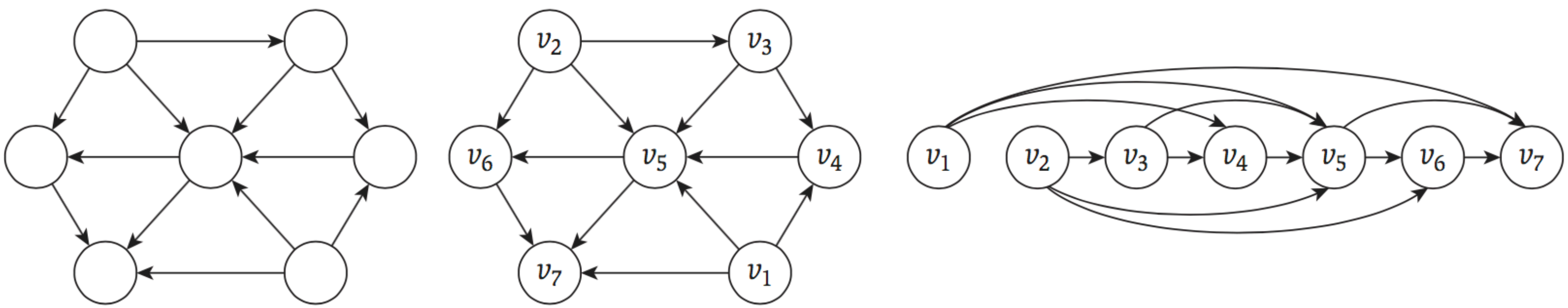
Directed Acyclic Graphs (DAGs)

- **DAG:** A directed graph with no directed cycles
- Can be much more complex than a forest



Directed Acyclic Graphs (DAGs)

- **DAG:** A directed graph with no directed cycles
- DAGs represent precedence relationships



- A topological ordering of a directed graph is a labeling of the nodes from $v_1, \ldots, v_n$ so that all edges go "forwards", that is $(v_i, v_j) \in E \Rightarrow j > i$
  - G has a topological ordering $\Rightarrow$ G is a DAG

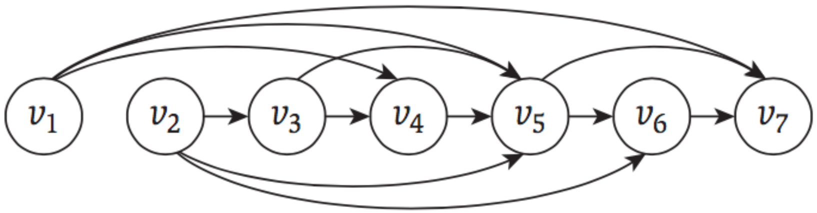
Directed Acyclic Graphs (DAGs)

- **Problem 1:** given a digraph G, is it a DAG?
- **Problem 2:** given a digraph G, can it be topologically ordered?

- **Thm:** G has a topological ordering $\Leftrightarrow$ G is a DAG
  - We will design one algorithm that either outputs a topological ordering or finds a directed cycle

Topological Ordering

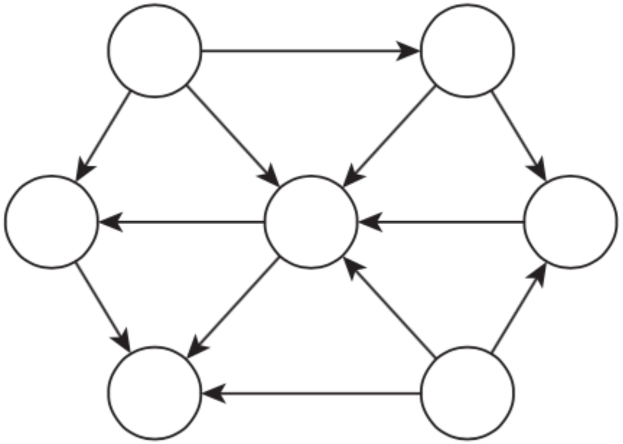
- **Observation:** the first node must have no in-edges



- **Observation:** In any DAG, there is always a node with no incoming edges

Topological Ordering

- **Fact:** In any DAG, there is a node with no incoming edges
- **Thm:** Every DAG has a topological ordering
- **Proof (Induction):**

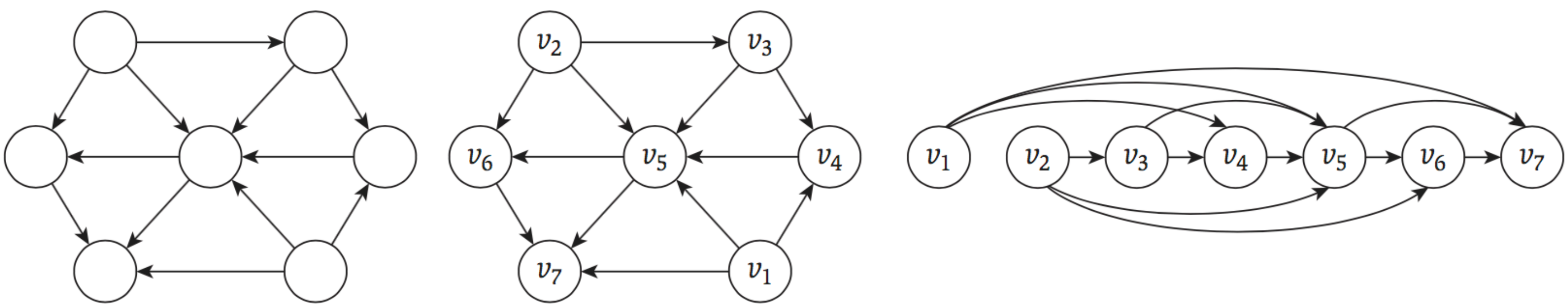


Fast Topological Ordering

- **Claim:** ordering nodes by decreasing postorder gives a topological ordering
- **Proof:**
  - A DAG has no backward edges
  - 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!

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 $(v_i, v_j) \in E \Longrightarrow j > i$



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