CS3000: Algorithms & Data
Paul Hand

Lecture 14:

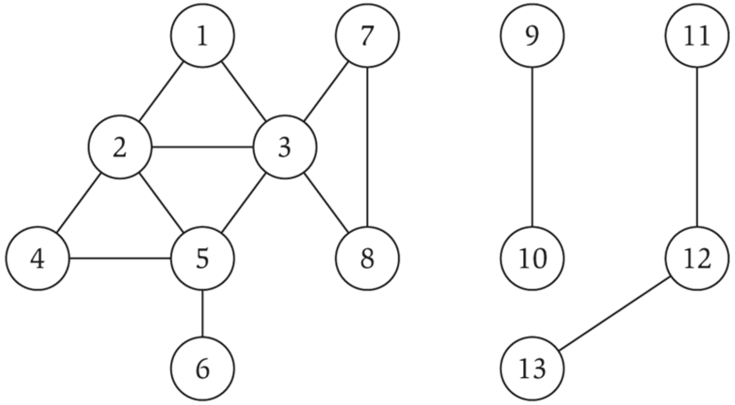
- Bipartite Graphs and 2-coloring
- Depth First Search

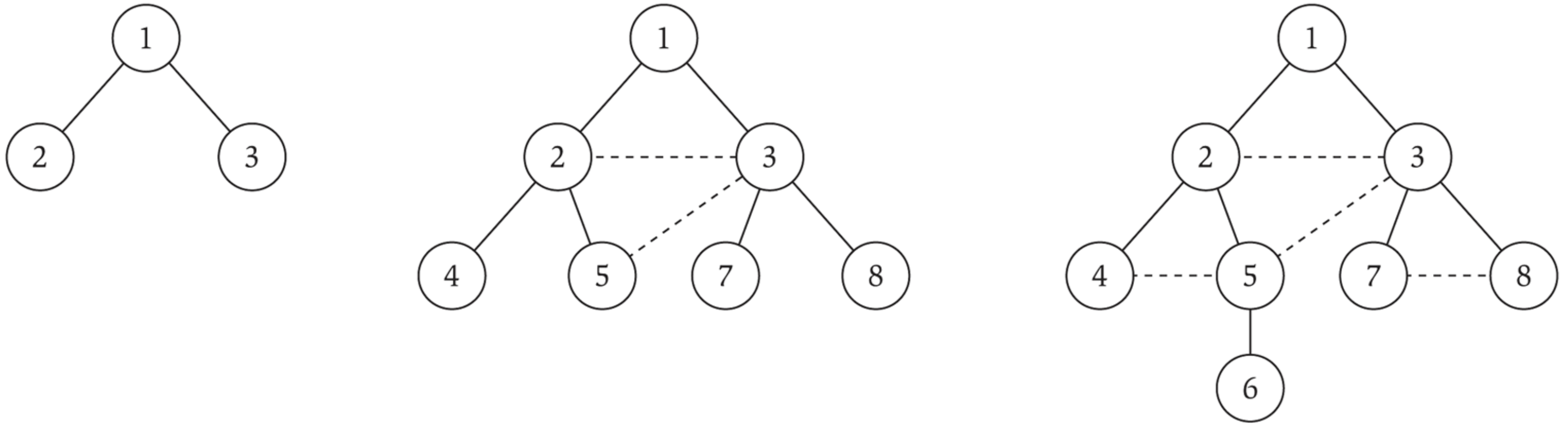
Mar 11, 2019

Recap: Graphs/BFS

Breadth-First Search (BFS)

- **Definition:** the distance between s, t is the number of edges on the shortest path from s to t
- **Thm:** BFS finds distances from s to other nodes
  - L_i contains all nodes at distance i from s
  - Nodes not in any layer are not reachable from s





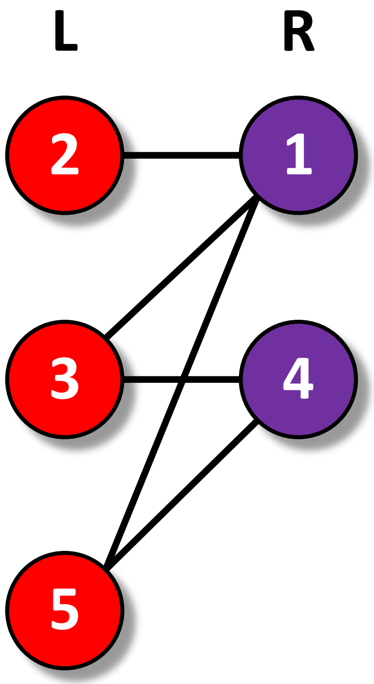
Bipartiteness / 2-Coloring

2-Coloring

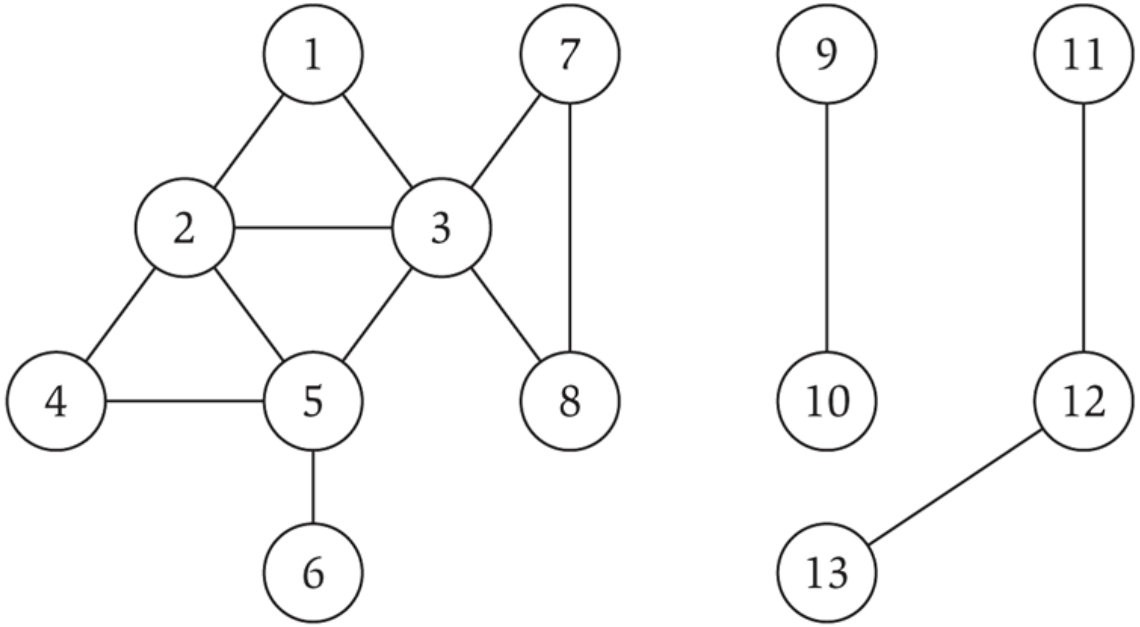
- **Problem:** Team Forming
  - Need to form two teams $\boldsymbol{R}$, $\boldsymbol{P}$
  - Some people don't want to be on the same team as certain other people
- **Input:** Undirected graph $G = (V, E)$
  - $(u, v) \in E$ means u, v wont be on the same team
- **Output:** Split V into two sets $\boldsymbol{R}$, $\boldsymbol{P}$ so that no pair in either set is connected by an edge

2-Coloring (Bipartiteness)

- **Equivalent Problem:** Is the graph G bipartite?
  - A graph G is bipartite if I can split V into two sets L and R such that all edges $(u, v) \in E$ go between L and R



Activity: Is the following graph bipartite?



Activity:

Give an example of a bipartite graph that is not connected

Suppose a graph of 10 nodes is bipartite. What is the maximum number of edges it can have?

Activity: Is the following graph bipartite?

All omitted entries are zero

A	1	2	3	4	5	6	7	8	9	10
1			1					1		
2				1	1					1
3	1					1		1		
4		1			1				1	
5		1		1					1	
6			1				1	1		
7						1		1		
8	1		1			1	1			
9				1	1					
10		1								

Designing an Algorithm to determine if a graph is bipartite

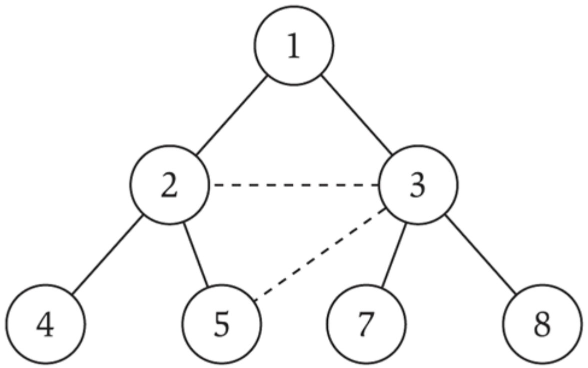
- **Key Fact:** If G contains a cycle of odd length, then G is not 2-colorable/bipartite

Designing the Algorithm

- **Idea for the algorithm:**
  - BFS the graph, coloring nodes as you find them
  - Color nodes in layer i **purple** if i even, **red** if i odd
  - See if you have succeeded or failed

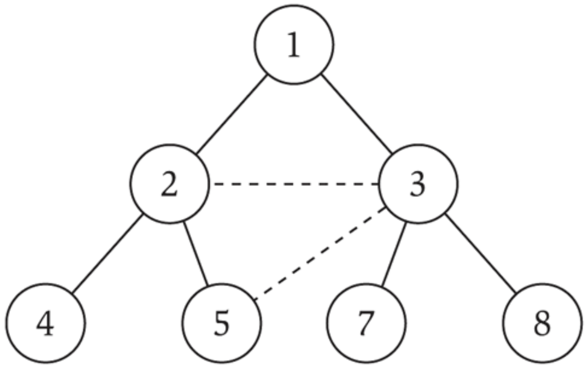
Designing the Algorithm

- **Claim:** If BFS 2-colored the graph successfully, the graph has been 2-colored successfully
- **Key Question:** Suppose you have not 2-colored the graph successfully, maybe someone else can do it?



Designing the Algorithm

- **Claim:** If BFS fails, then G contains an odd cycle
  - If G contains an odd cycle then G can't be 2-colored!



Depth-First Search (DFS)

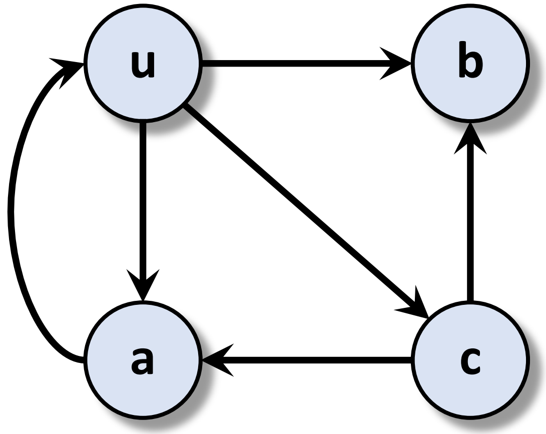
Exploring a Graph

- **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

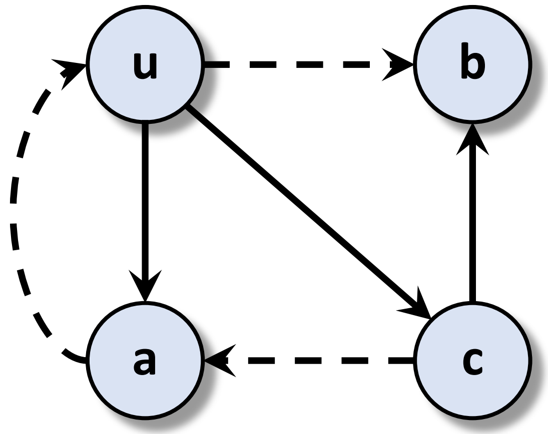
Depth-First Search

```
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)
```



Depth-First Search

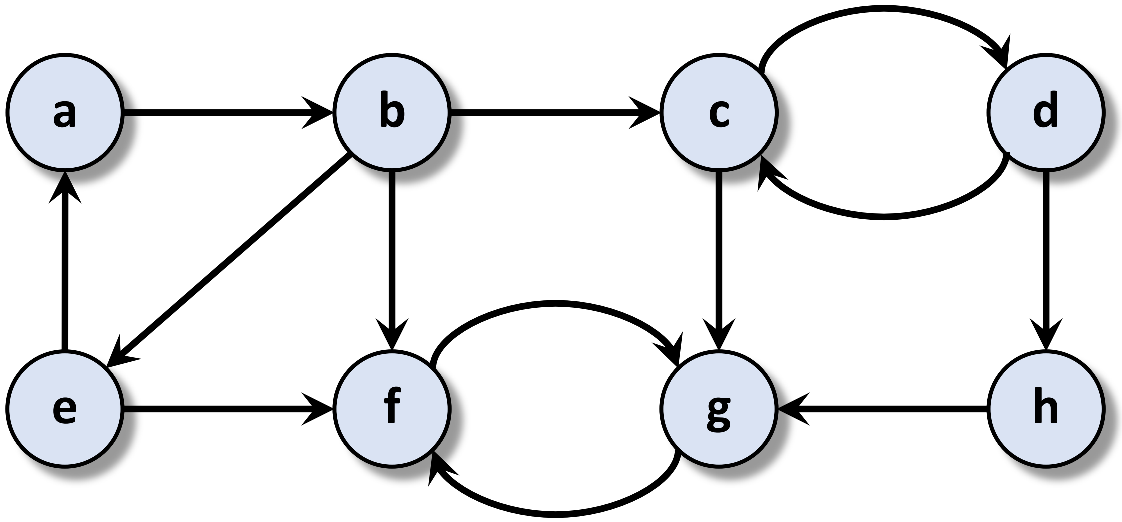
- **Fact:** The parent-child edges form a (directed) tree
- **Each edge has a type:**
  - **Tree edges:** $(u, a), (u, c), (c, b)$
    - These are the edges that explore new nodes
  - **Forward edges:** (u, b)
    - Ancestor to descendant
  - **Backward edges:** (a, u)
    - Descendant to ancestor
 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



Activity

- **Each edge has a type:**
  - **Tree edges:** $(u, a), (u, c), (c, b)$
    - 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

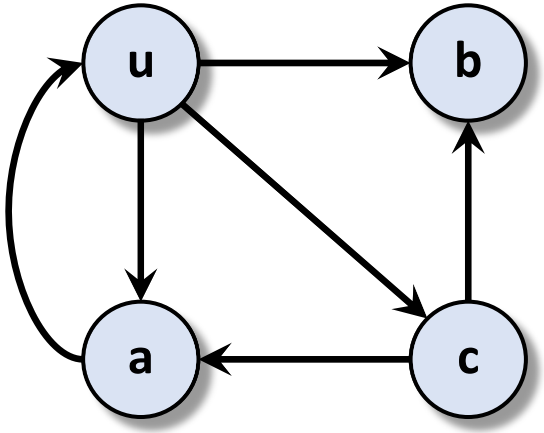
- DFS this graph starting from node a
  - Search in alphabetical order
  - Label edges as { tree , forward , backward , cross}



Pre-Ordering

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

```
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)
```



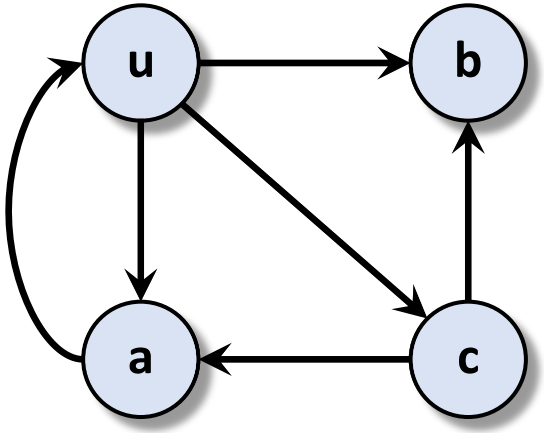
Vertex	Pre-Order

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

Post-Ordering

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

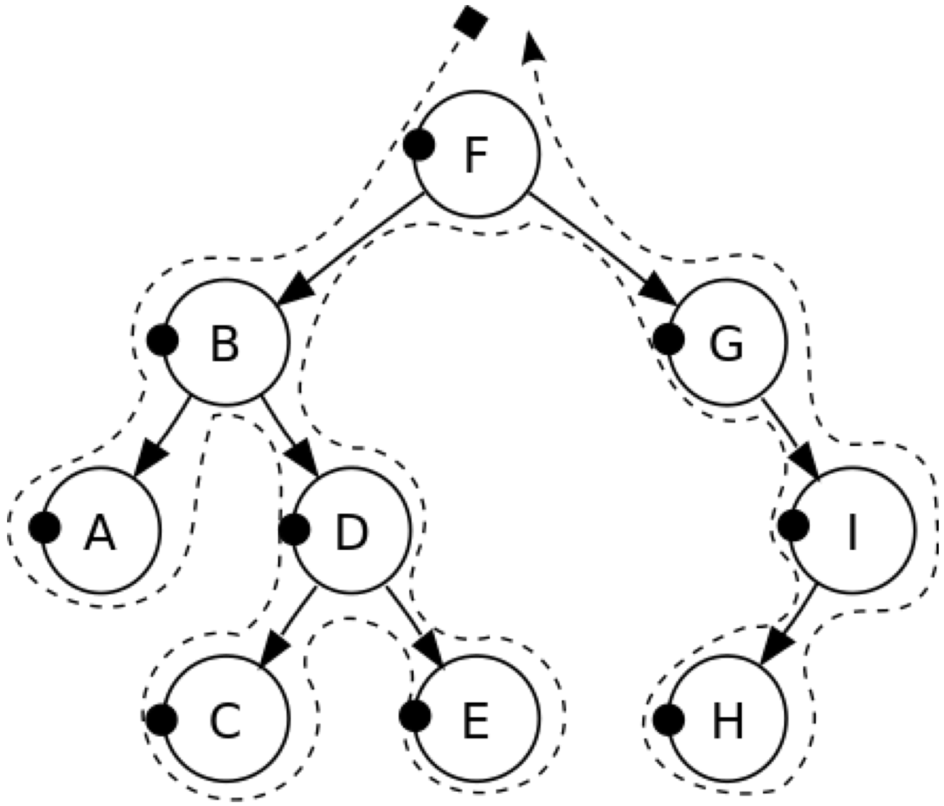
```
G = (V,E) is a graph
explored[u] = 0 ∀u

DFS(u):
  explored[u] = 1

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

  post-visit(u)
```



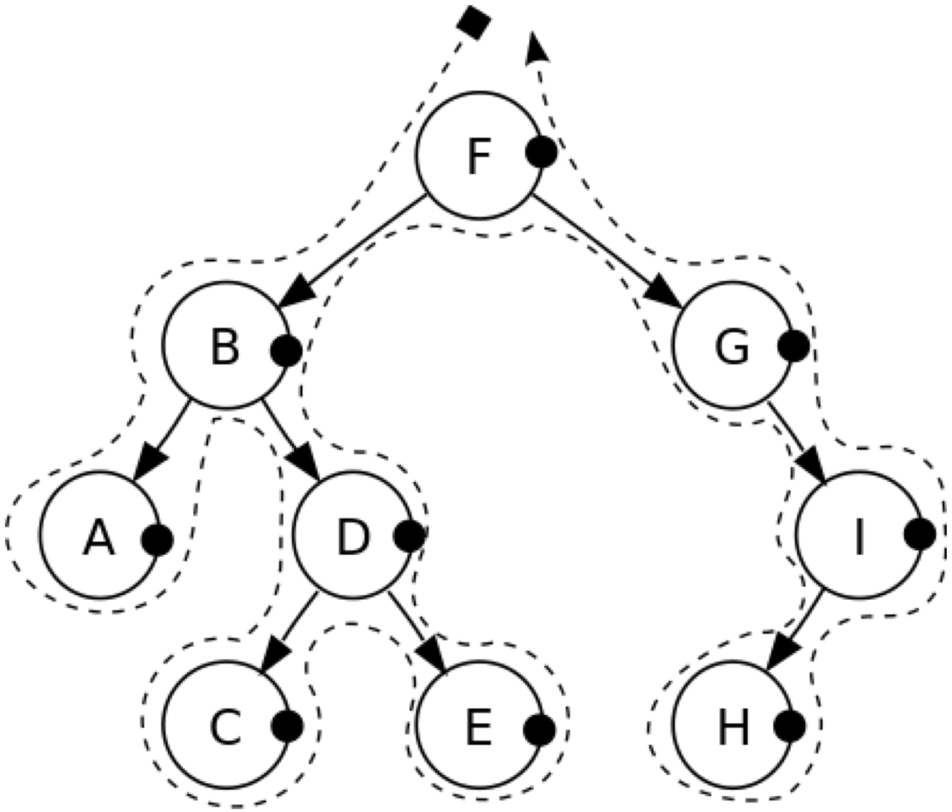
Vertex	Post-Order

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

Preorder versus postorder



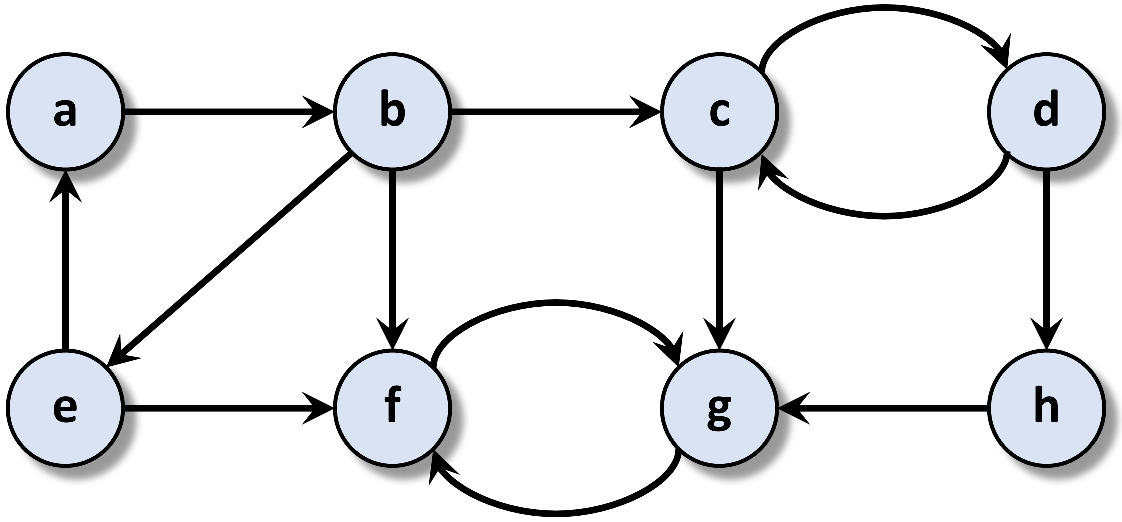
Pre-order: F, B, A, D, C, E, G, I, H.



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

Activity

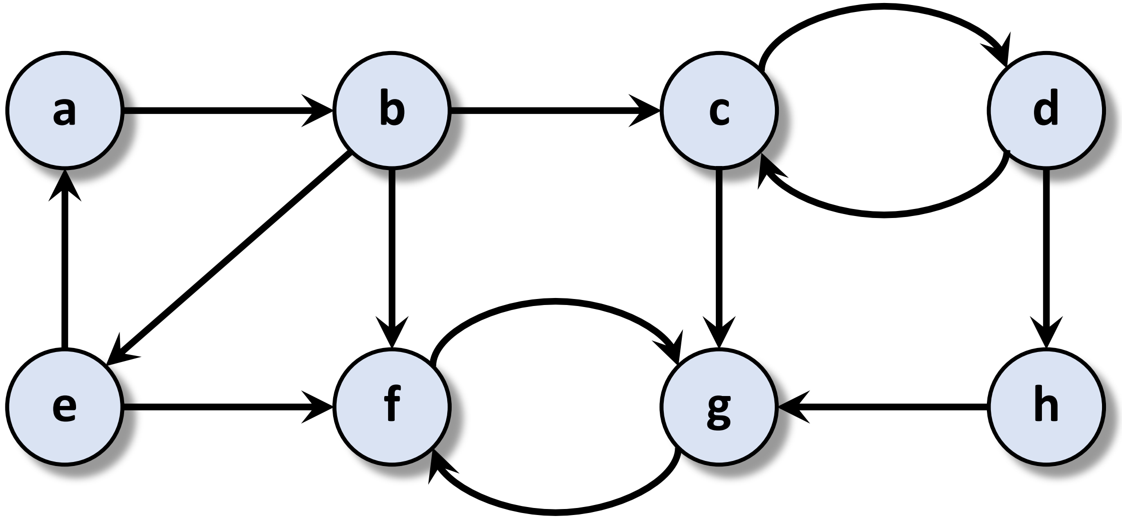
- Compute the **post-order** of this graph
  - DFS from $\boldsymbol{a}$, search in alphabetical order



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

Ask the Audience

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



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