Revised: 7/8/2024

Graphs I

Michael C. Hackett

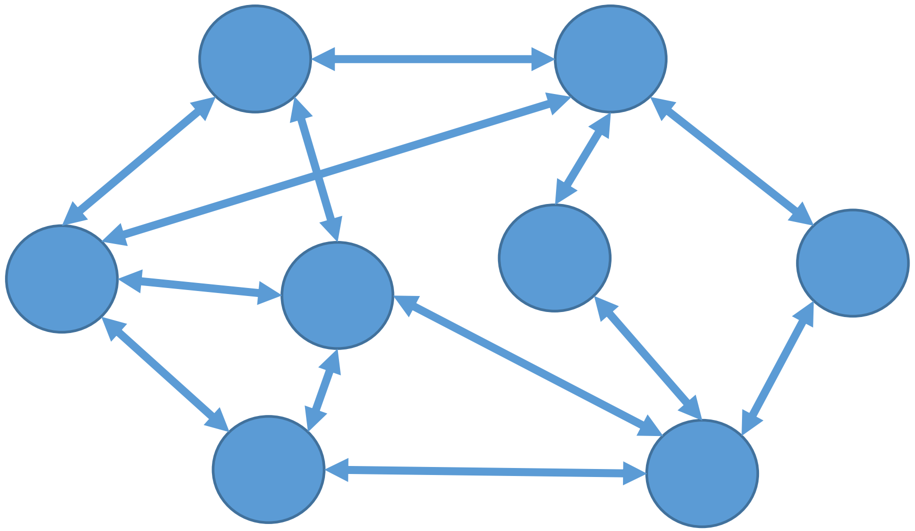
Assistant Professor, Computer Science

Lecture Topics

- Graph Terminology
- Implementing Graphs
  - Adjacency Lists
    - Adding an edge
    - Removing an edge
    - Checking if an edge exists
  - Adjacency Matrices
    - Adding an edge
    - Removing an edge
    - Checking if an edge exists
  - Complexity Comparison
- Directed Graphs
  - Adding an edge
  - Removing an edge
  - Checking if an edge exists
  - Complexity Comparison

Graphs

- A **graph** is a non-linear data structure, where each node is connected by one or more edges.



Graphs

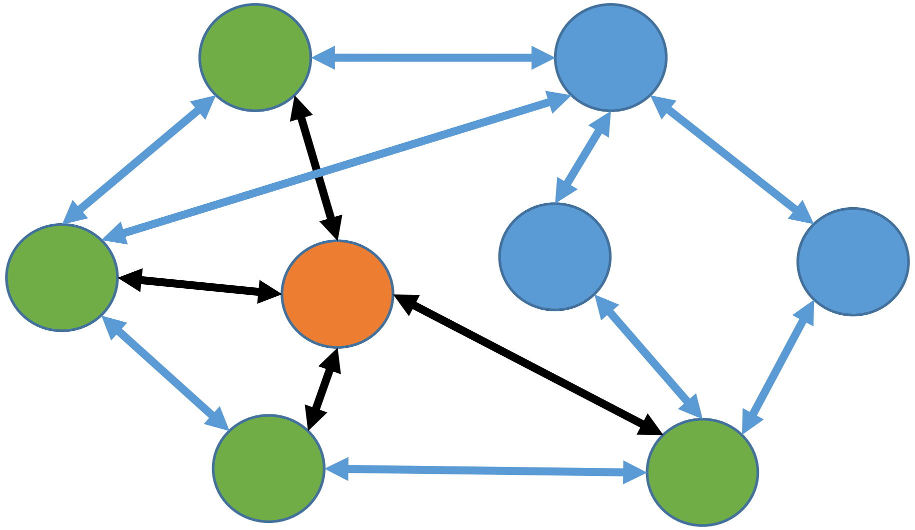
- There is no “root” node.

- Each node has one or more edges.
  - A node could connect to every other node, or connect to only one.

- No parent/child relationships

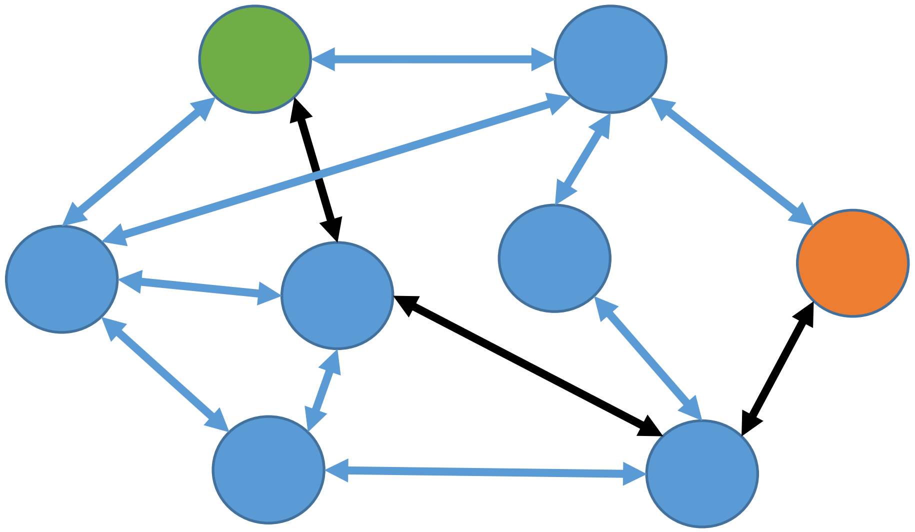
Graphs

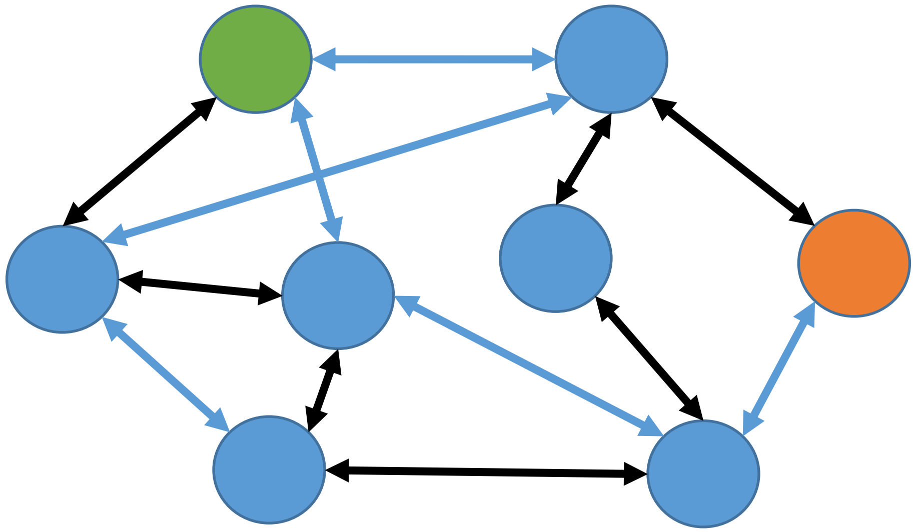
- Two nodes are **adjacent** if they are connected by an edge.
  - “Neighbors”
  - The green nodes below are adjacent to the orange node



Graphs

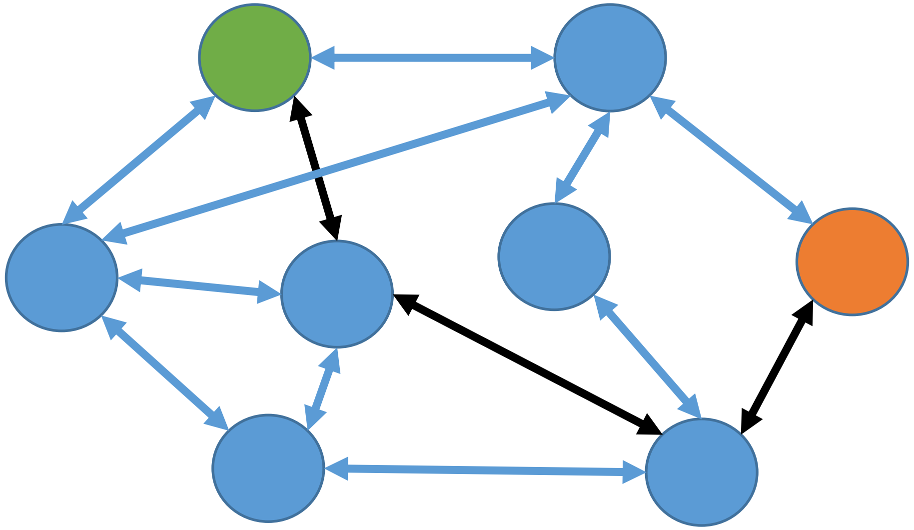
- A **path** is the sequence of edges between two nodes in the structure.
  - The green node below is the starting node and the orange node is the ending node; Black edges are a possible path.
  - More than one path may exist.



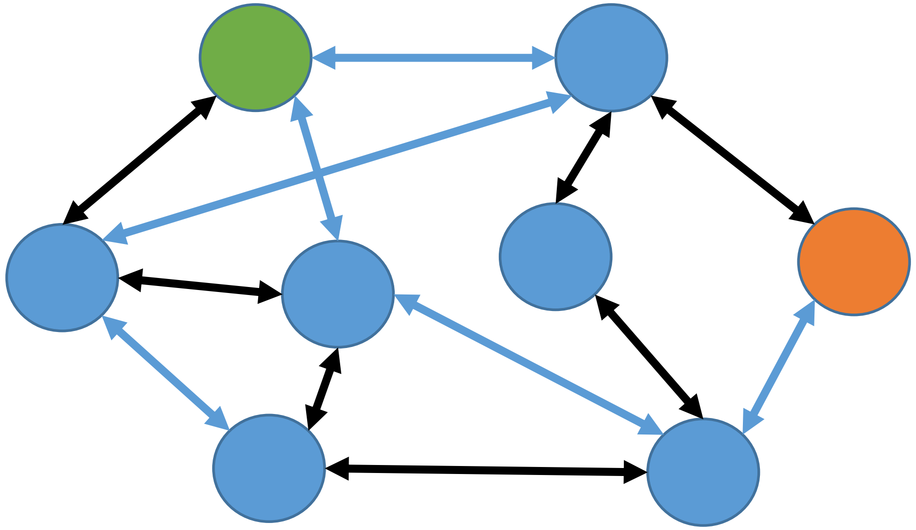


Graphs

- The **path length** is the number of edges in a path.



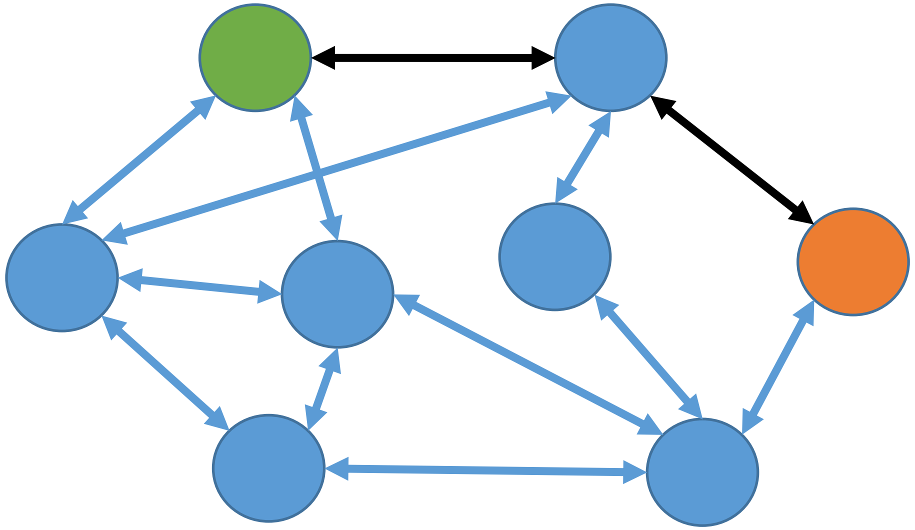
Path length = 3



Path length = 7

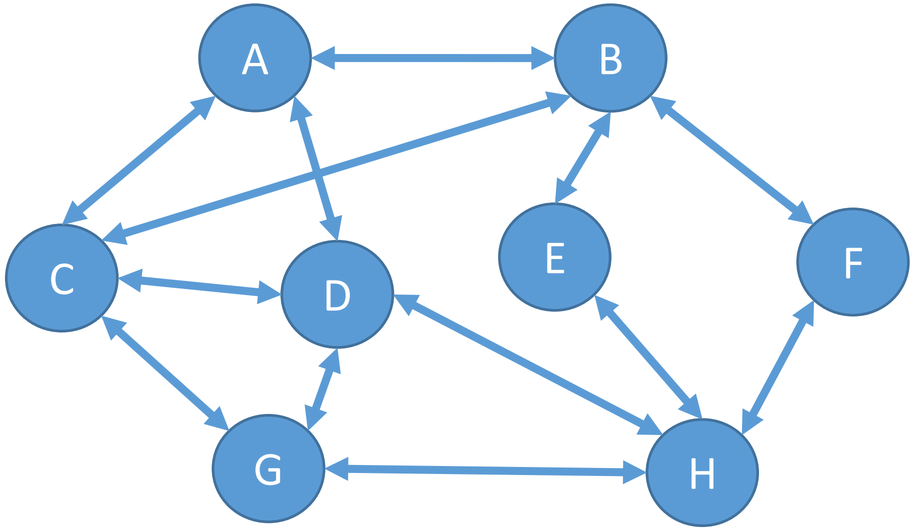
Graphs

- The **distance** between two nodes is the shortest path length between them.
  - The distance between the two nodes in this example graph is 2.



Adjacency Lists

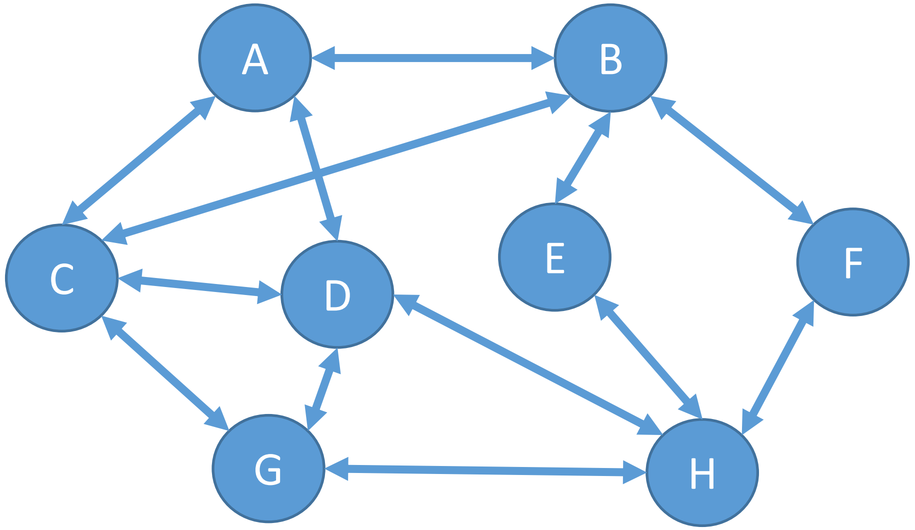
- One way to implement a graph is using an **adjacency list**.
  - Each node corresponds to a List of adjacent nodes



Node	Adjacent Nodes
A	B, C, D
B	A, C, E, F
C	A, B, D, G
D	A, C, G, H
E	B, H
F	B, H
G	C, D, H
H	D, E, F, G

Adjacency Lists – Adding an Edge

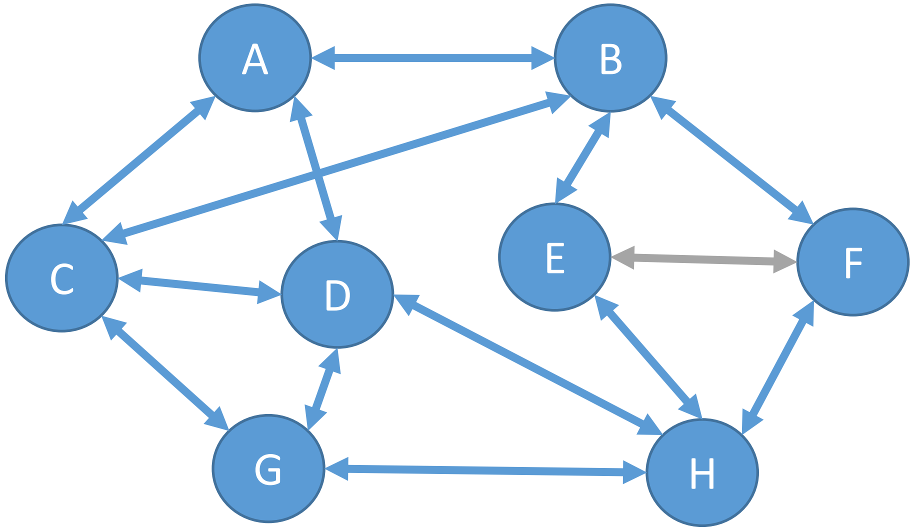
- To add an edge between two nodes, each node is placed in the other’s adjacency list



Node	Adjacent Nodes
A	B, C, D
B	A, C, E, F
C	A, B, D, G
D	A, C, G, H
E	B, H
F	B, H
G	C, D, H
H	D, E, F, G

Adjacency Lists – Adding an Edge

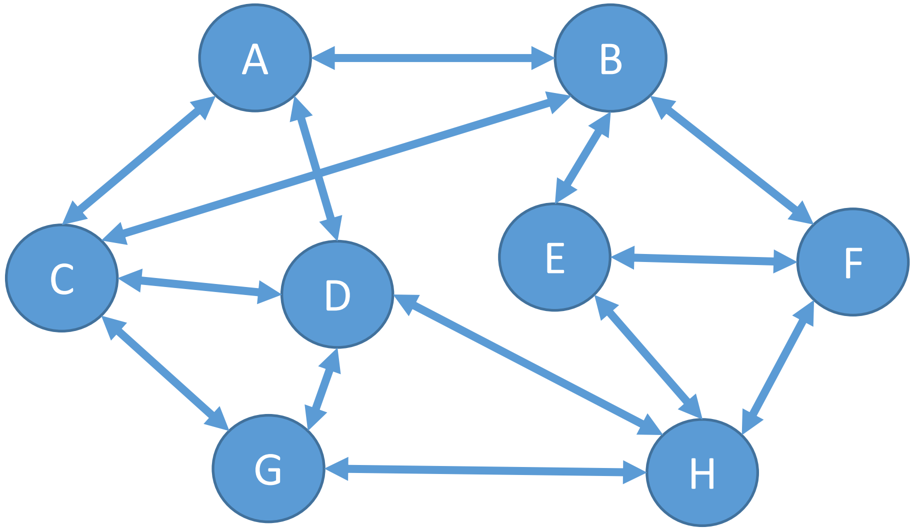
- Adding an edge between E and F



Node	Adjacent Nodes
A	B, C, D
B	A, C, E, F
C	A, B, D, G
D	A, C, G, H
E	B, H
F	B, H
G	C, D, H
H	D, E, F, G

Adjacency Lists – Adding an Edge

- E is added to F's adjacency list
- F is added to E's adjacency list



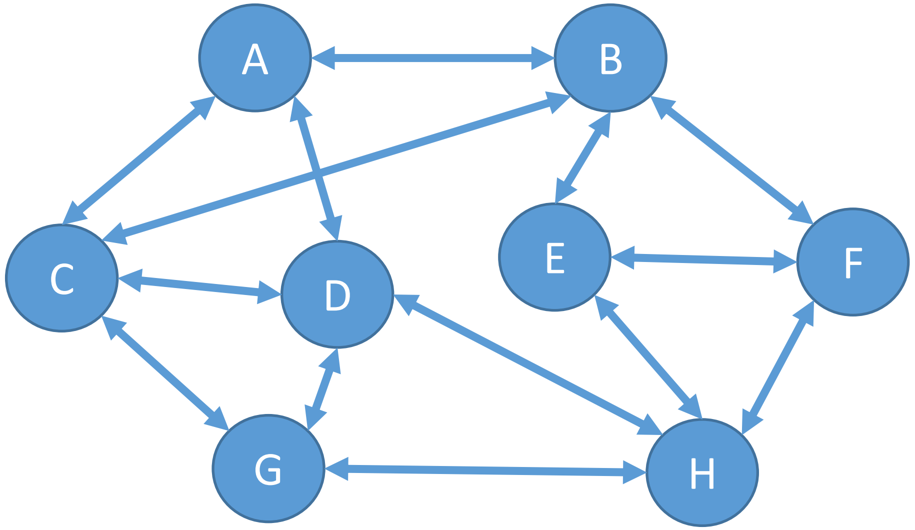
Node	Adjacent Nodes
A	B, C, D
B	A, C, E, F
C	A, B, D, G
D	A, C, G, H
E	B, H, **F**
F	B, H, **E**
G	C, D, H
H	D, E, F, G

Adjacency Lists – Adding an Edge

- Time Complexity (Adding an edge): **O(1)**
  - Adding an edge simply appends the node/edge to a node's adjacency list
  - Adding/appending to a List takes constant time

Adjacency Lists – Removing an Edge

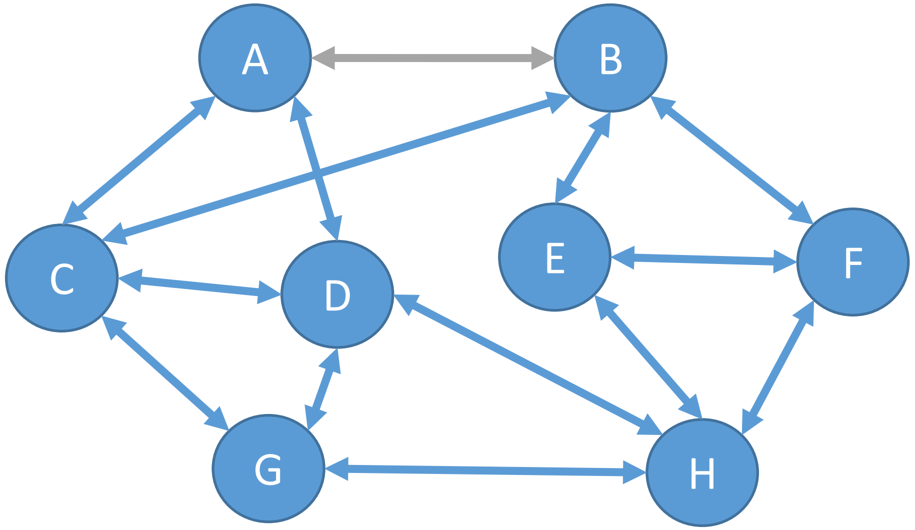
- To remove an edge between two nodes, each node is removed from the other's adjacency list



Node	Adjacent Nodes
A	B, C, D
B	A, C, E, F
C	A, B, D, G
D	A, C, G, H
E	B, H, F
F	B, H, E
G	C, D, H
H	D, E, F, G

Adjacency Lists – Removing an Edge

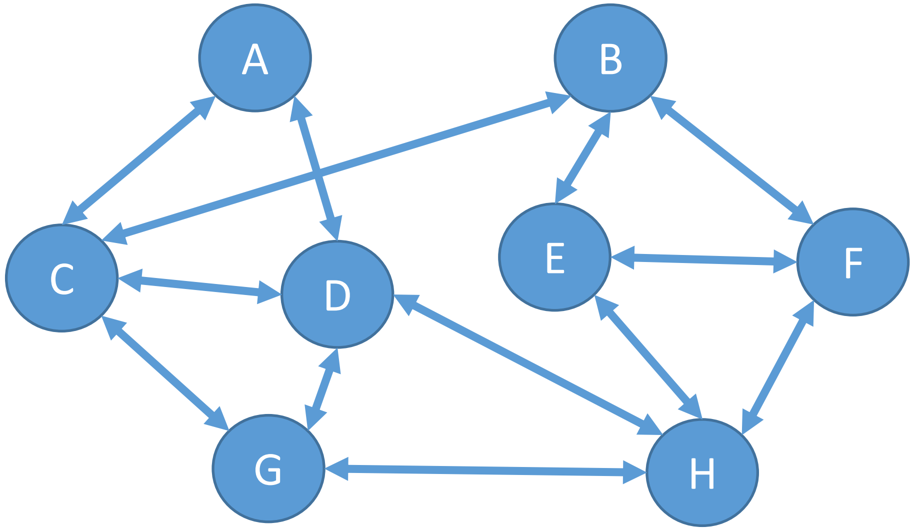
- Removing the edge between A and B



Node	Adjacent Nodes
A	B, C, D
B	A, C, E, F
C	A, B, D, G
D	A, C, G, H
E	B, H, F
F	B, H, E
G	C, D, H
H	D, E, F, G

Adjacency Lists – Removing an Edge

- B is removed from A’s adjacency list
- A is removed from B’s adjacency list



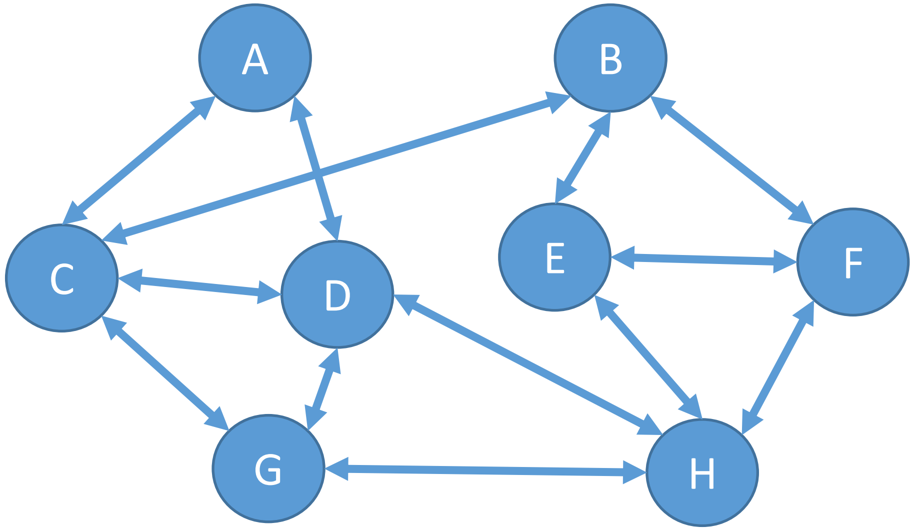
Node	Adjacent Nodes
A	~~B~~, C, D
B	~~A~~, C, E, F
C	A, B, D, G
D	A, C, G, H
E	B, H, F
F	B, H, E
G	C, D, H
H	D, E, F, G

Adjacency Lists – Removing an Edge

- Time Complexity (Removing an edge): **$O(e_1 + e_2)$**
  - e_1 = Number of nodes/edges in the first node's adjacency list
  - e_2 = Number of nodes/edges in the second node's adjacency list
  - The first node's adjacency list is searched (linearly) to find and remove the second node
  - The second node's adjacency list is searched (linearly) to find and remove the first node

Adjacency Lists – Checking for an edge

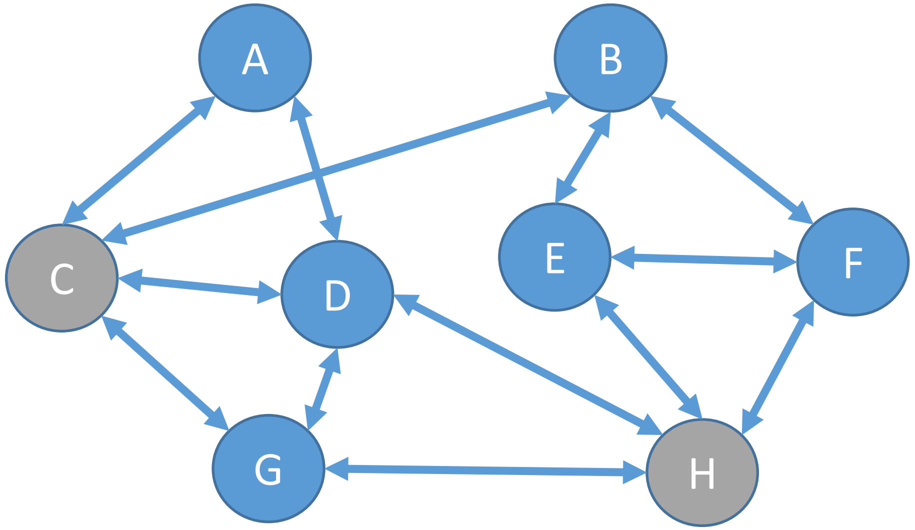
- To check if an edge exists between two nodes, the node's adjacency list is checked for the other node



Node	Adjacent Nodes
A	C, D
B	C, E, F
C	A, B, D, G
D	A, C, G, H
E	B, H, F
F	B, H, E
G	C, D, H
H	D, E, F, G

Adjacency Lists – Checking for an edge

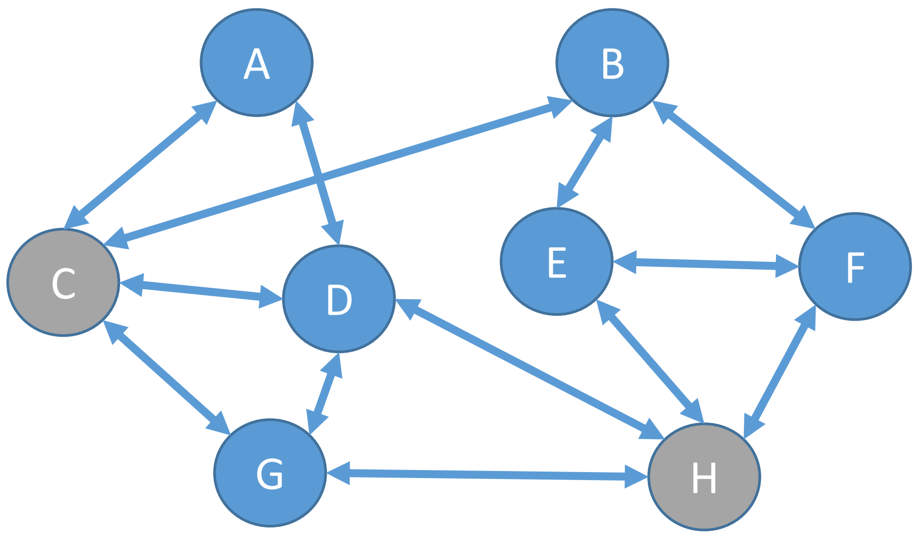
- Checking for an edge between C and H



Node	Adjacent Nodes
A	C, D
B	C, E, F
C	A, B, D, G
D	A, C, G, H
E	B, H, F
F	B, H, E
G	C, D, H
H	D, E, F, G

Adjacency Lists – Checking for an edge

- H is not in C's adjacency list
  - Alternatively could have checked H's adjacency list for C
  - Does not imply a *path* doesn't exist



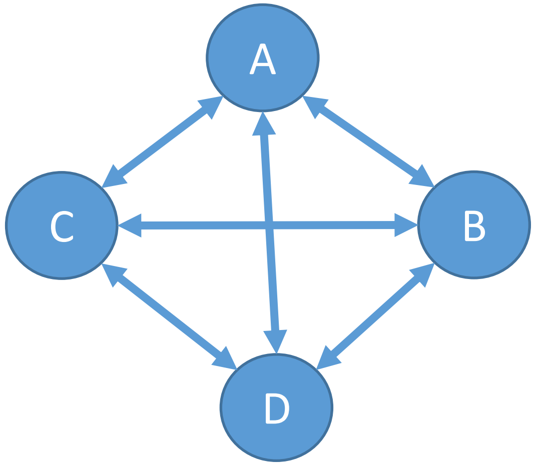
Node	Adjacent Nodes
A	C, D
B	C, E, F
C	**A, B, D, G**
D	A, C, G, H
E	B, H, F
F	B, H, E
G	C, D, H
H	D, E, F, G

Adjacency Lists – Checking for an edge

- Time Complexity (Checking for an edge): **O(e)**
  - e = Number of nodes/edges in the node's adjacency list
  - Determining one node's adjacency to a second node is verified by iterating through the adjacent nodes in either node's adjacency list.

Adjacency Lists – Space Complexity

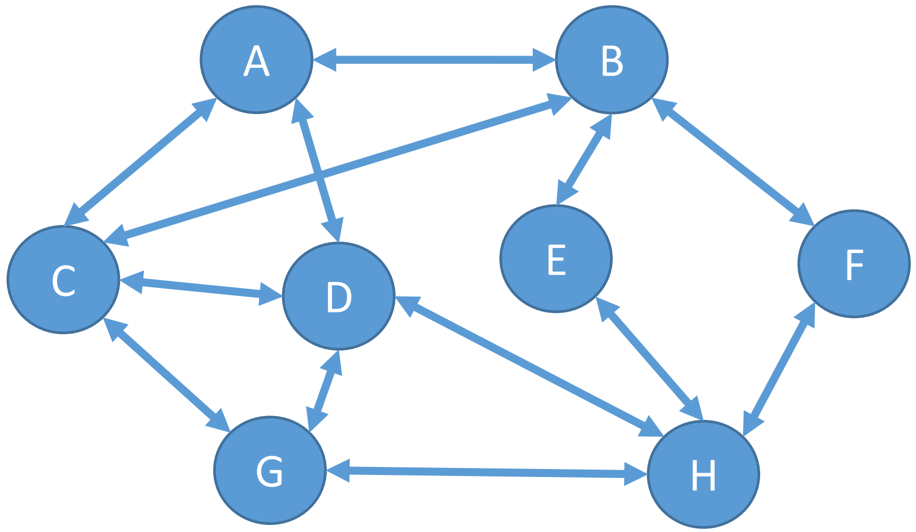
- (Worst Case) Space Complexity: O(V+E)
  - V = Total number of vertices (Nodes)
  - E = Total number of edges (Adjacent Nodes)



Node	Adjacent Nodes
A	B, C, D
B	A, C, D
C	A, B, D
D	A, B, C

Adjacency Matrices

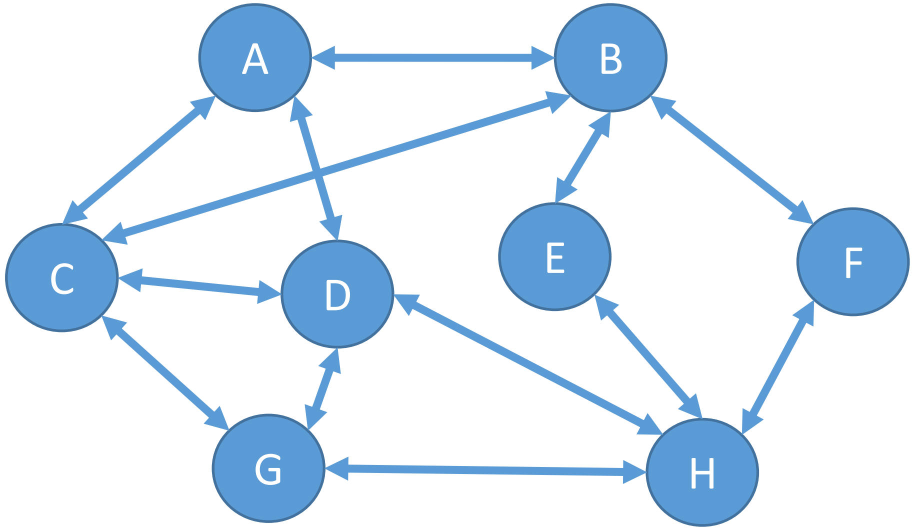
- Another way to implement a graph is using an **adjacency matrix**.
  - 2-D Array
  - 1 if an edge exists, 0 if not



	A	B	C	D	E	F	G	H
A		1	1	1				
B	1		1		1	1		
C	1	1		1			1	
D	1		1				1	1
E		1						1
F		1						1
G			1	1				1
H				1	1	1	1	

Adjacency Matrices – Adding an Edge

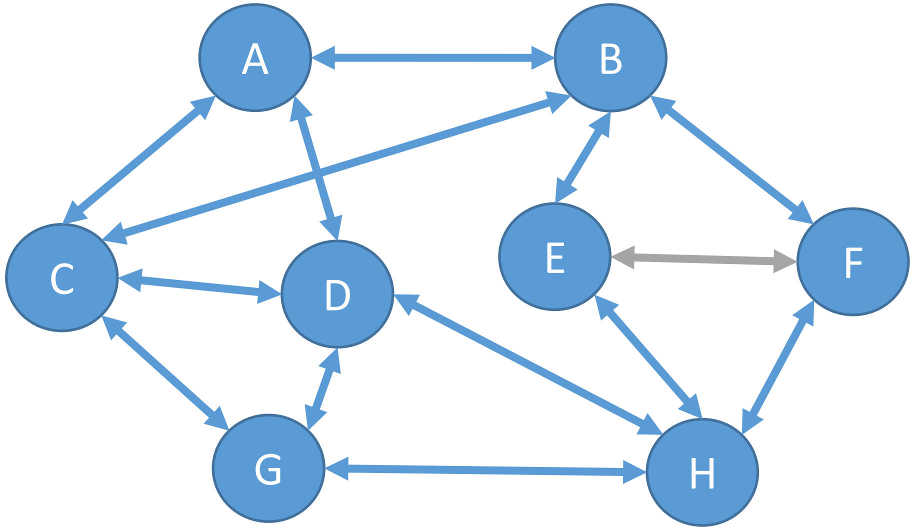
- To add an edge between two nodes, a 1 is placed in the matrix



	A	B	C	D	E	F	G	H
A		1	1	1				
B	1		1		1	1		
C	1	1		1			1	
D	1		1				1	1
E		1						1
F		1						1
G			1	1				1
H				1	1	1	1	

Adjacency Matrices – Adding an Edge

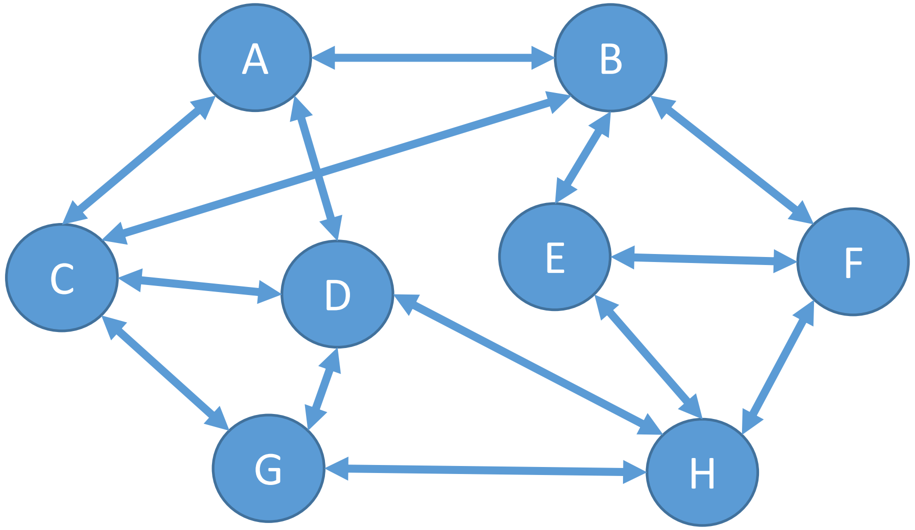
- Adding an edge between E and F



	A	B	C	D	E	F	G	H
A		1	1	1				
B	1		1		1	1		
C	1	1		1			1	
D	1		1				1	1
E		1						1
F		1						1
G			1	1				1
H				1	1	1	1	

Adjacency Matrices – Adding an Edge

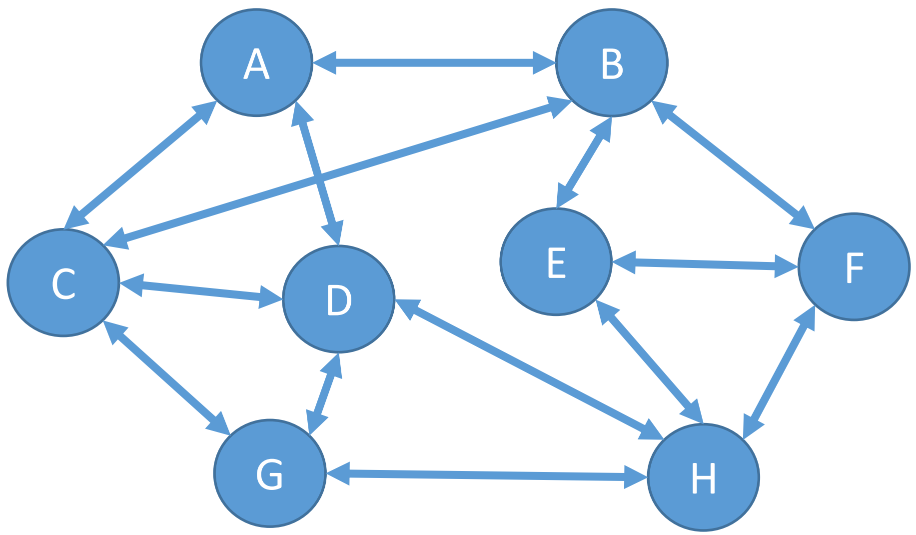
- A 1 is placed in the corresponding locations in the matrix



	A	B	C	D	E	F	G	H
A		1	1	1				
B	1		1		1	1		
C	1	1		1			1	
D	1		1				1	1
E		1				**1**		1
F		1			**1**			1
G			1	1				1
H				1	1	1	1	

Adjacency Matrices – Removing an Edge

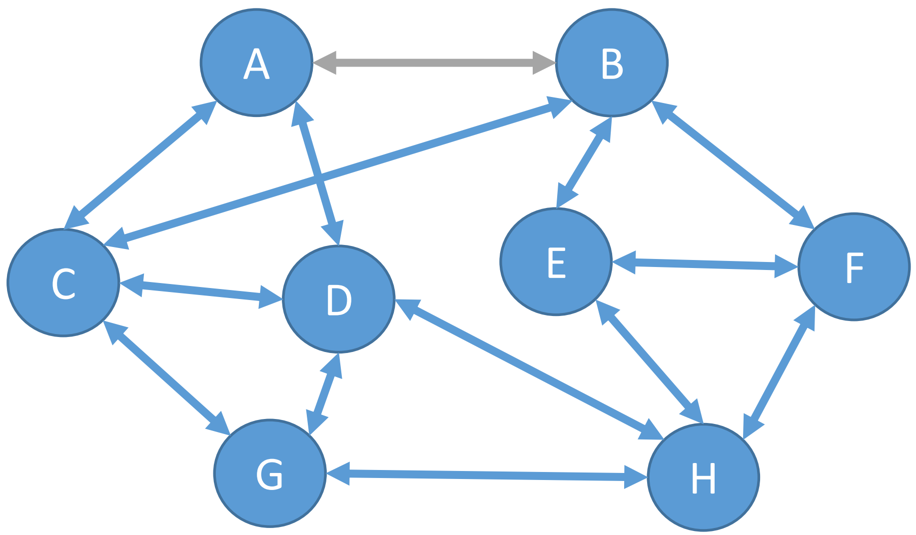
- To remove an edge between two nodes, a 0 is placed in the corresponding locations in the matrix
  - In this example, blank cells represent 0



	A	B	C	D	E	F	G	H
A		1	1	1				
B	1		1		1	1		
C	1	1		1			1	
D	1		1				1	1
E		1				1		1
F		1			1			1
G			1	1				1
H				1	1	1	1	

Adjacency Matrices – Removing an Edge

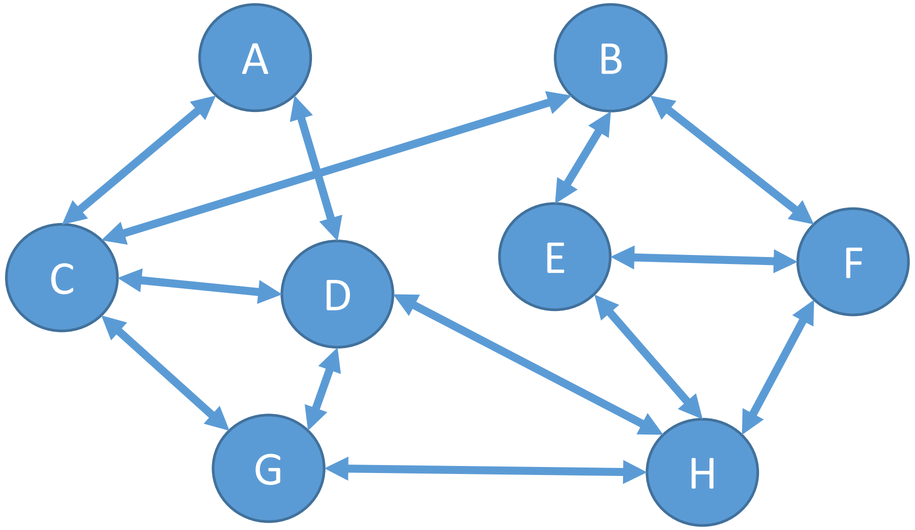
- Removing the edge between A and B



	A	B	C	D	E	F	G	H
A		**1**	1	1				
B	**1**		1		1	1		
C	1	1		1			1	
D	1		1				1	1
E		1				1		1
F		1			1			1
G			1	1				1
H				1	1	1	1	

Adjacency Matrices – Removing an Edge

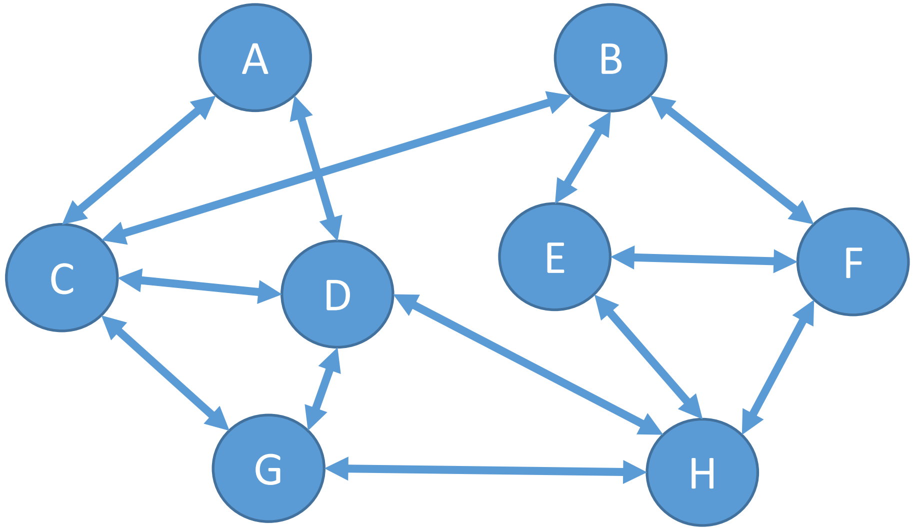
- A 0 is placed in the corresponding locations in the matrix



	A	B	C	D	E	F	G	H
A		**0**	1	1				
B	**0**		1		1	1		
C	1	1		1			1	
D	1		1				1	1
E		1				1		1
F		1			1			1
G			1	1				1
H				1	1	1	1	

Adjacency Matrices – Checking for an edge

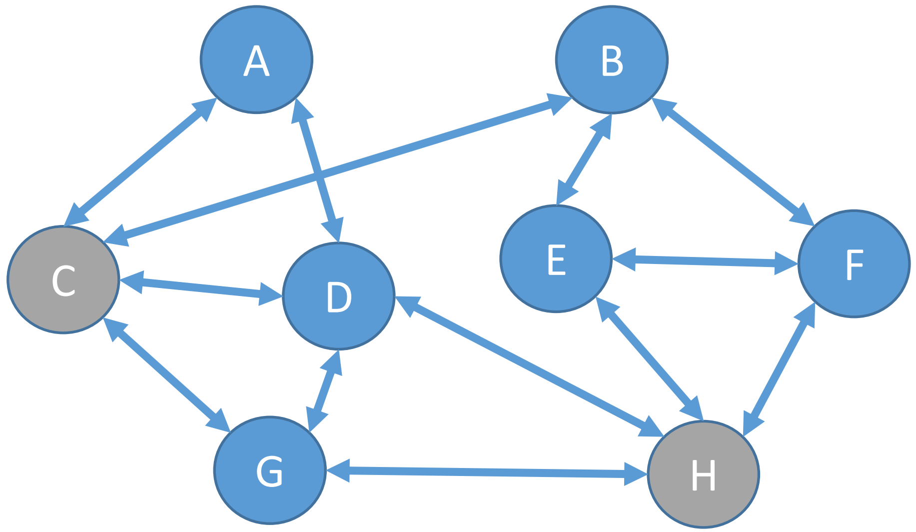
- To check if an edge exists between two nodes, that location in the matrix is checked for a 1



	A	B	C	D	E	F	G	H
A			1	1				
B			1		1	1		
C	1	1		1			1	
D	1		1				1	1
E		1				1		1
F		1			1			1
G			1	1				1
H				1	1	1	1	

Adjacency Matrices – Checking for an edge

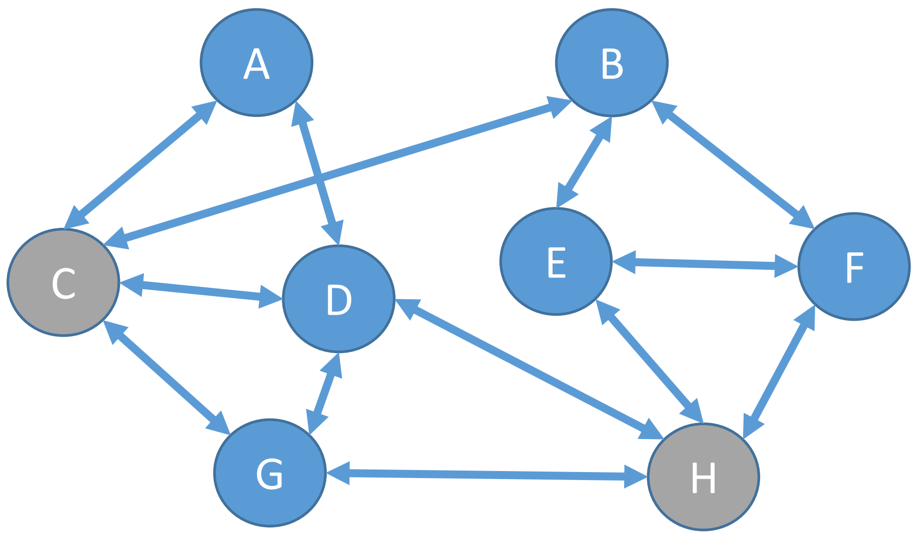
- Checking if C is adjacent to H



	A	B	C	D	E	F	G	H
A			1	1				
B			1		1	1		
C	1	1		1			1	
D	1		1				1	1
E		1				1		1
F		1			1			1
G			1	1				1
H				1	1	1	1	

Adjacency Matrices – Checking for an edge

- There is not a 1 in [C][H]
  - Alternatively could have checked [H][C]
  - Does not imply a *path* doesn't exist



	A	B	C	D	E	F	G	H
A			1	1				
B			1		1	1		
C	1	1		1			1	**0**
D	1		1				1	1
E		1				1		1
F		1			1			1
G			1	1				1
H			**0**	1	1	1	1	

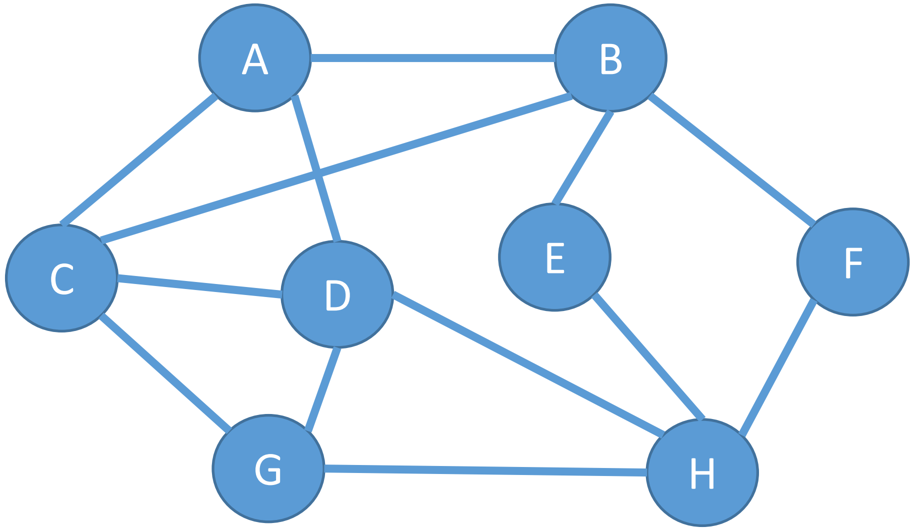
Adjacency Matrices

- Time Complexity (Adding an edge): **O(1)**
- Time Complexity (Removing an edge): **O(1)**
- Time Complexity (Checking for an edge): **O(1)**

- Array access takes constant time

Adjacency Matrices

- Space Complexity: always $O(V^2)$
  - V = Number of vertices



	A	B	C	D	E	F	G	H
A		1	1	1				
B	1		1		1	1		
C	1	1		1			1	
D	1		1				1	1
E		1						1
F		1						1
G			1	1				1
H				1	1	1	1	

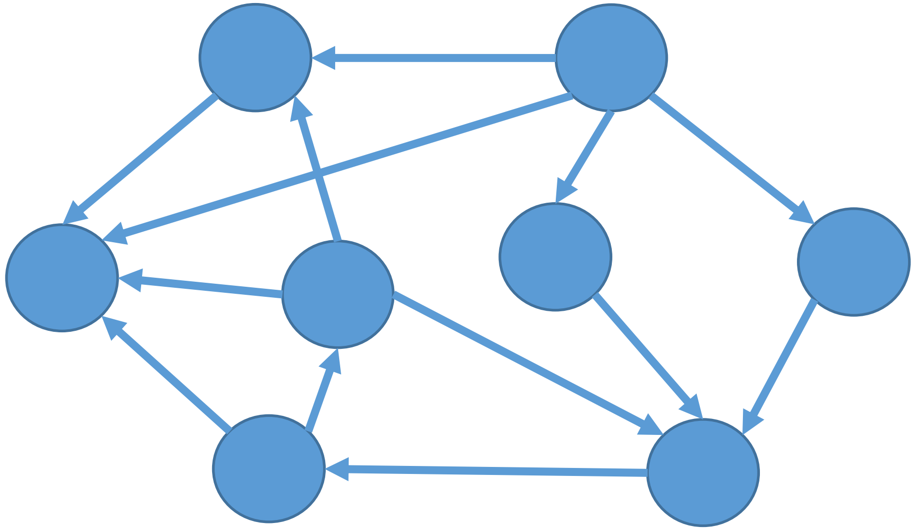
Complexity Comparison

Graph Implementation	Add Edge	Remove Edge	Check Edge	Space
Adjacency List	$O(1)$	$O(e_1 + e_2)$	$O(e)$	$O(V + E)$
Adjacency Matrix	$O(1)$	$O(1)$	$O(1)$	$O(V^2)$

Green - Constant
Orange - Linear
Red - Polynomial

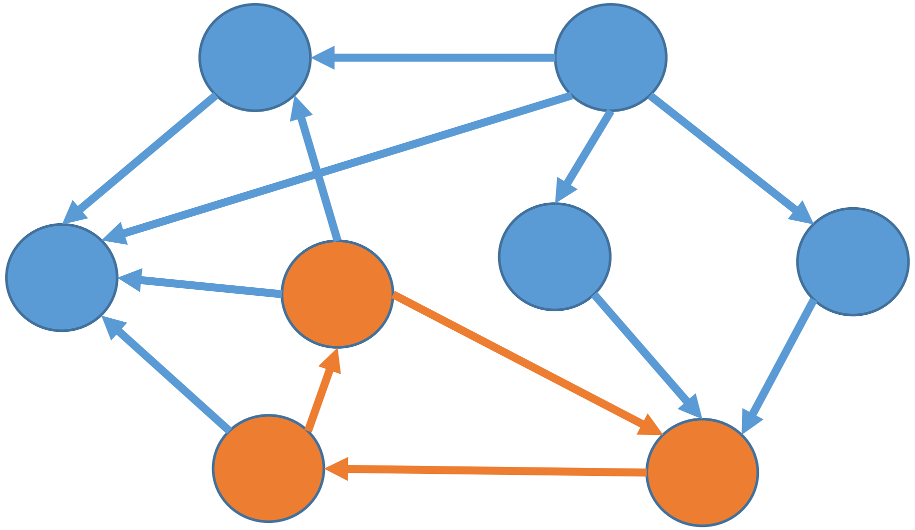
Directed Graphs

- A **directed graph** (or **digraph**) is a graph where each node is connected via <u>one-way</u> edges.
  - Previously, we were using a *bi-directional* graph



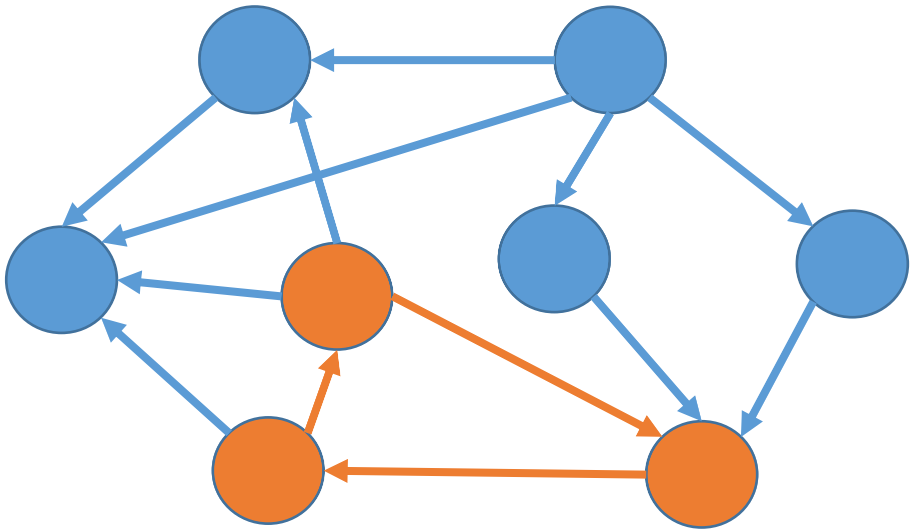
Directed Graphs

- A **cycle** is a path that can begin and end at the same node in a directed graph.

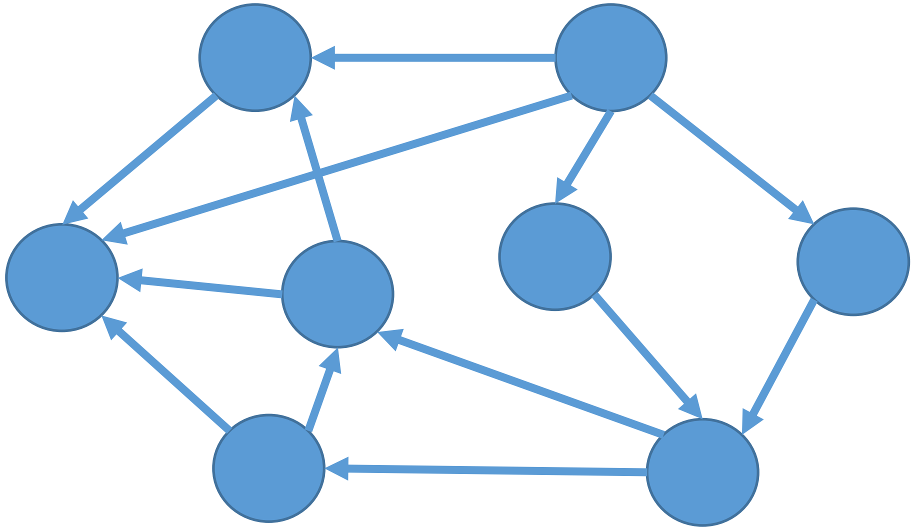


Directed Graphs

- A graph that contains a cycle is **cyclic**
  - Bi-directional graphs are inherently cyclic
- A graph that does not contain a cycle is **acyclic**



Cyclic



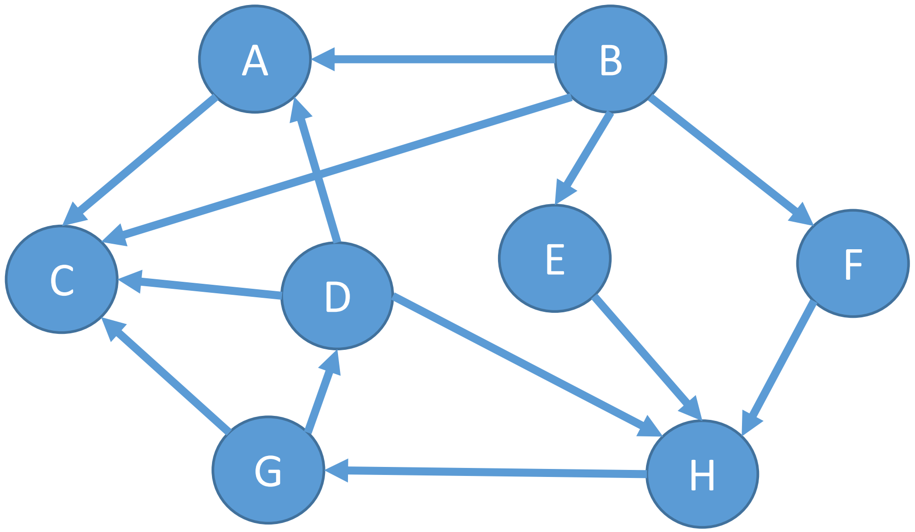
Acyclic

Directed Graphs

- Digraphs can be implemented using adjacency lists or adjacency matrices
- Processes for adding, removing, and checking edges in digraphs are mostly the same as for bi-directional graphs
- The remaining slides only show processes for a digraph using an adjacency list

Directed Graphs – Adding an Edge

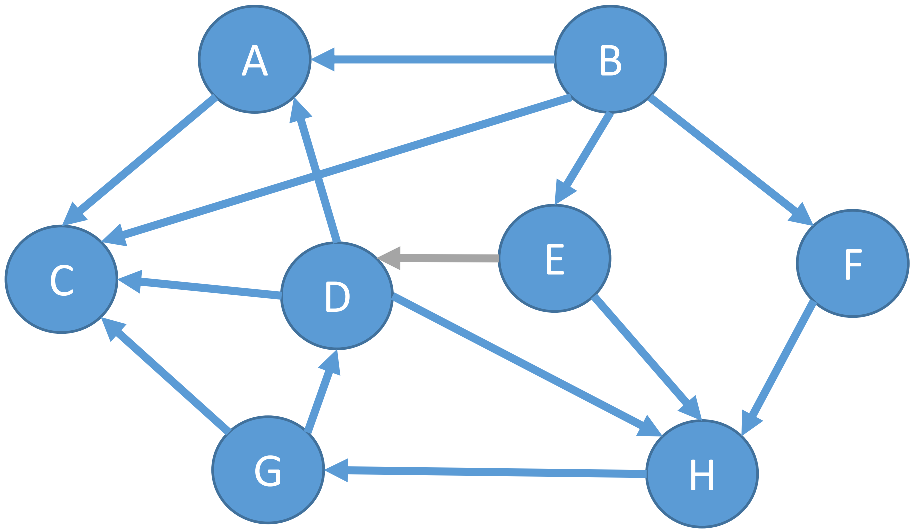
- To add an edge between two nodes, the end node is placed in the starting node's adjacency list



Node	Adjacent Nodes
A	C
B	A, C, E, F
C	
D	A, C, H
E	H
F	H
G	C, D
H	G

Directed Graphs – Adding an Edge

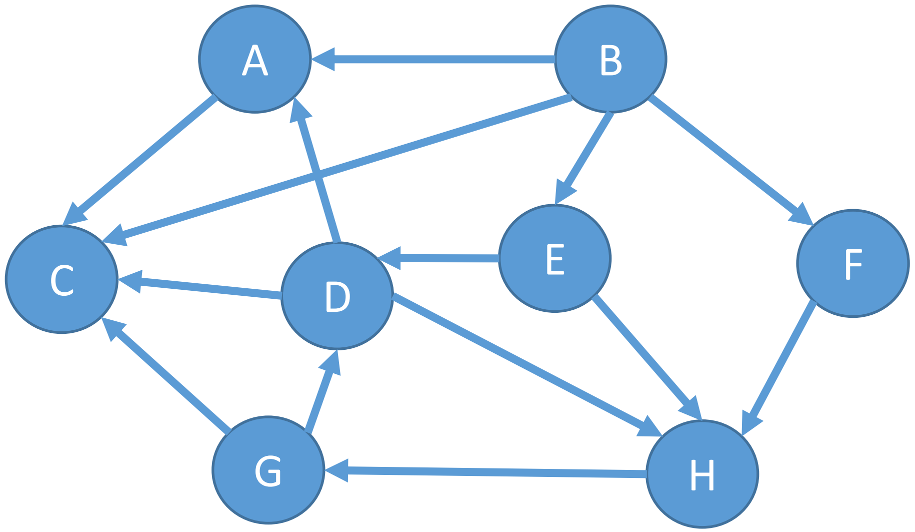
- Adding an edge from E to D



Node	Adjacent Nodes
A	C
B	A, C, E, F
C	
D	A, C, H
E	H
F	H
G	C, D
H	G

Directed Graphs – Adding an Edge

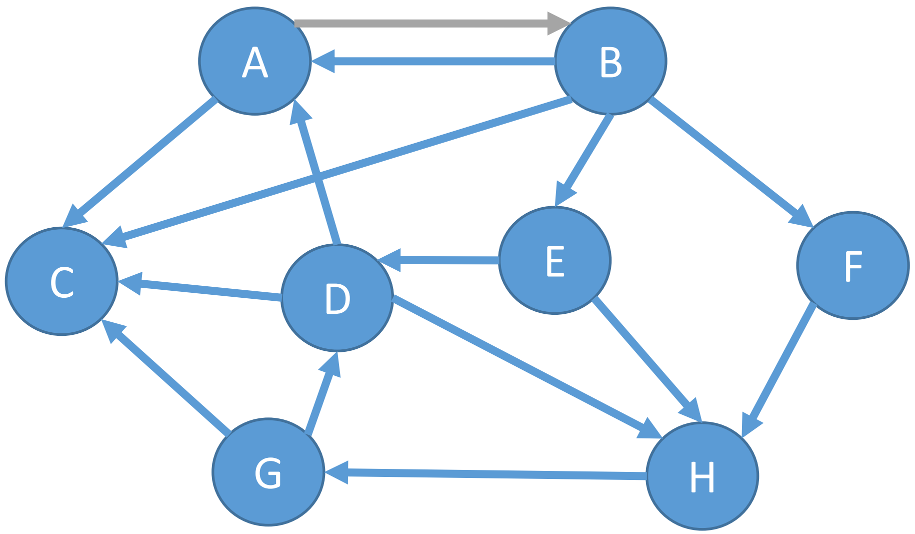
- D is added to E's adjacency list



Node	Adjacent Nodes
A	C
B	A, C, E, F
C	
D	A, C, H
E	H, **D**
F	H
G	C, D
H	G

Directed Graphs – Adding an Edge

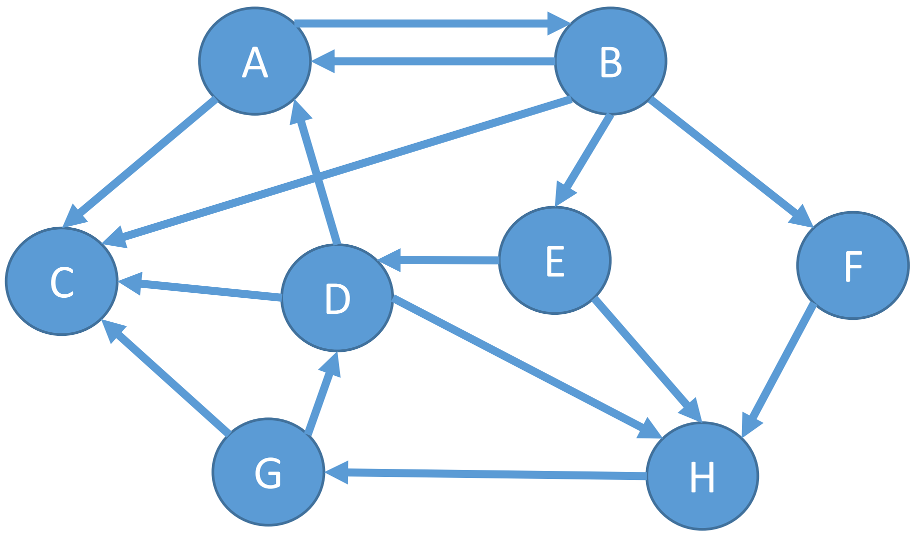
- Digraphs can have bi-directional edges
  - Adding an edge from A to B



Node	Adjacent Nodes
A	C
B	A, C, E, F
C	
D	A, C, H
E	H, D
F	H
G	C, D
H	G

Directed Graphs – Adding an Edge

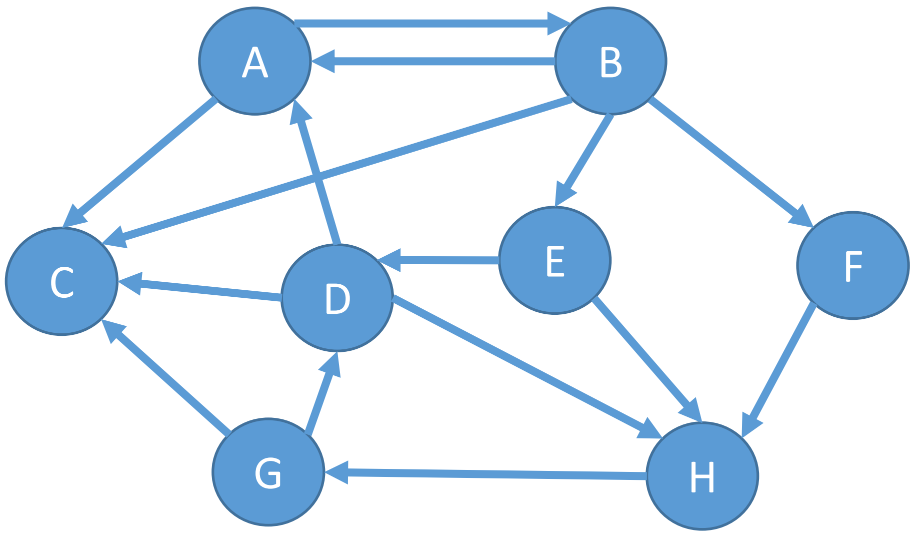
- B is added to A's adjacency list



Node	Adjacent Nodes
A	C, **B**
B	A, C, E, F
C	
D	A, C, H
E	H, D
F	H
G	C, D
H	G

Directed Graphs – Removing an Edge

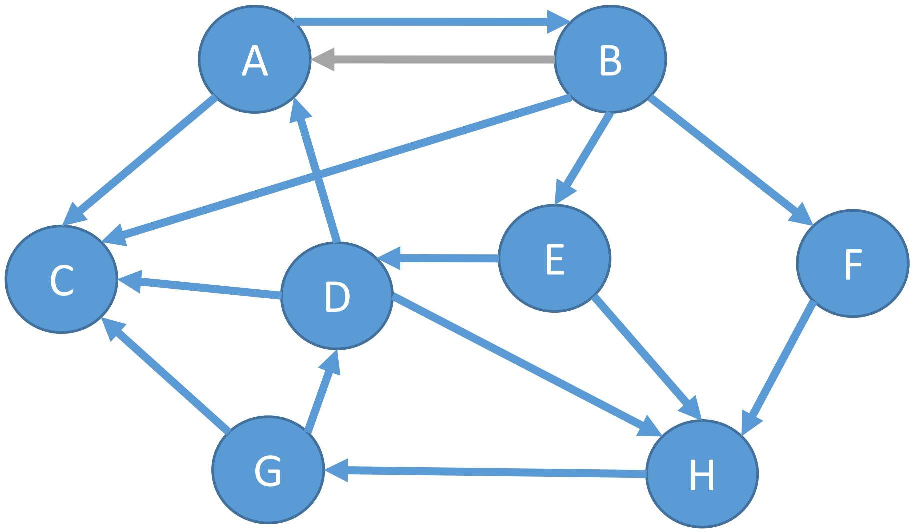
- To remove an edge, the end node is removed from the starting node's adjacency list



Node	Adjacent Nodes
A	C, B
B	A, C, E, F
C	
D	A, C, H
E	H, D
F	H
G	C, D
H	G

Directed Graphs – Removing an Edge

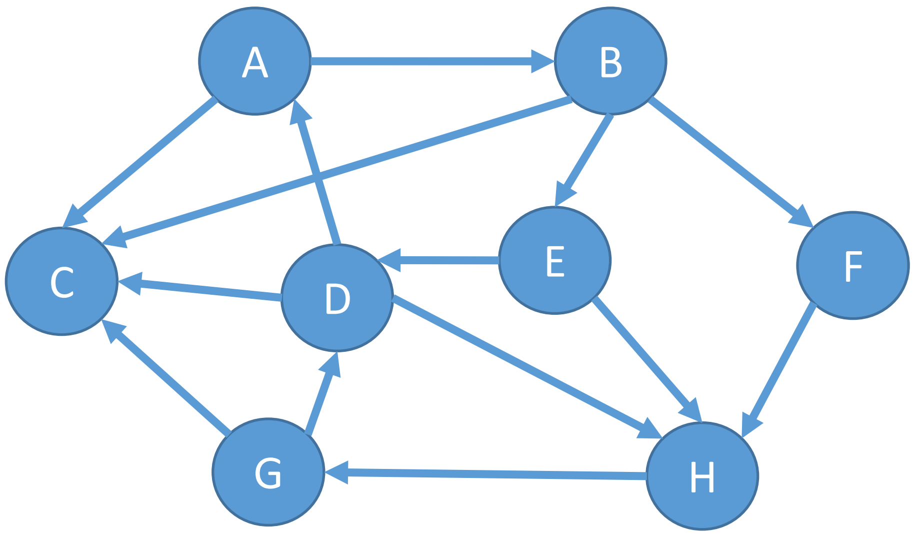
- Removing the edge from B to A



Node	Adjacent Nodes
A	C, B
B	~~A~~, C, E, F
C	
D	A, C, H
E	H, D
F	H
G	C, D
H	G

Directed Graphs – Removing an Edge

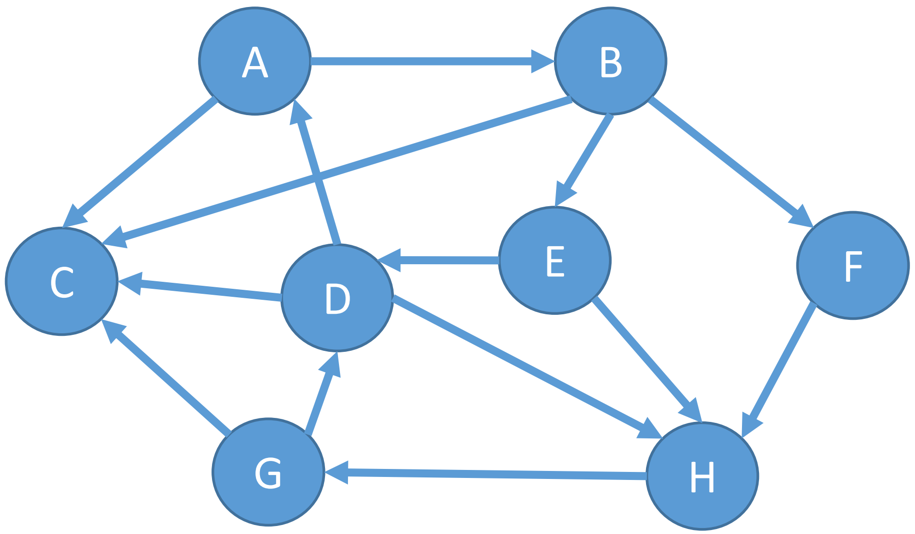
- A is removed from B's adjacency list



Node	Adjacent Nodes
A	C, B
B	C, E, F
C	
D	A, C, H
E	H, D
F	H
G	C, D
H	G

Directed Graphs – Checking for an edge

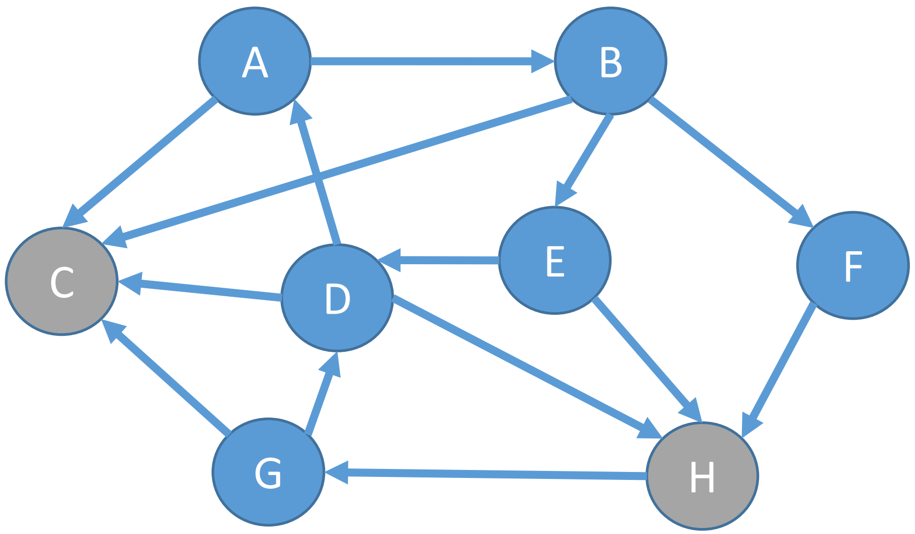
- To check if an edge exists between two nodes, the starting node's adjacency list is checked for the other node



Node	Adjacent Nodes
A	C, B
B	C, E, F
C	
D	A, C, H
E	H, D
F	H
G	C, D
H	G

Directed Graphs – Checking for an edge

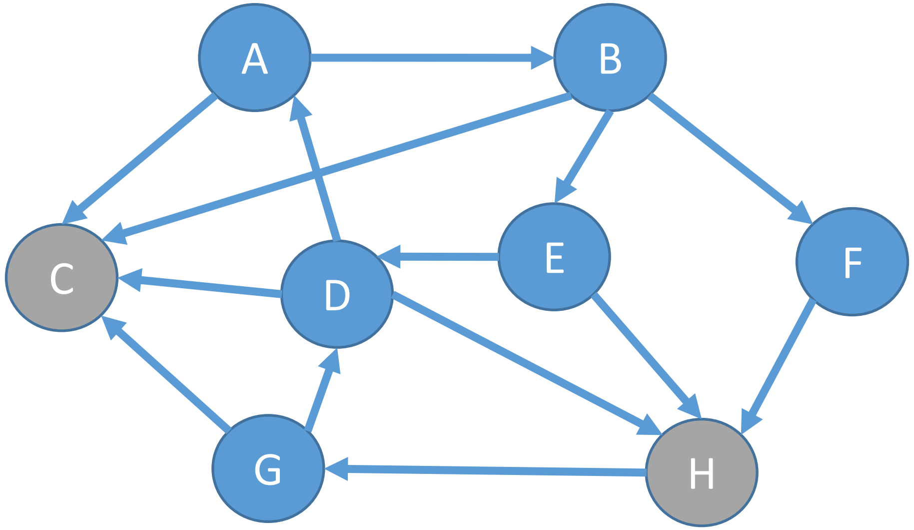
- Checking if H is adjacent to C



Node	Adjacent Nodes
A	C, B
B	C, E, F
C	
D	A, C, H
E	H, D
F	H
G	C, D
H	G

Directed Graphs – Checking for an edge

- C does not exist in H’s adjacency list
  - Since this is a digraph, we could not alternatively check to see if H is in C’s adjacency list



Node	Adjacent Nodes
A	C, B
B	C, E, F
C	
D	A, C, H
E	H, D
F	H
G	C, D
H	**G**

Directed Graphs – Complexity Comparison

Graph Implementation	Add Edge	Remove Edge	Check Edge	Space
Adjacency List	O(1)	$O(e_1 + e_2)$	O(e)	O(V + E)
Adjacency Matrix	O(1)	O(1)	O(1)	$O(V^2)$
Adjacency List (Digraph)	O(1)	O(e)	O(e)	O(V + E)
Adjacency Matrix (Digraph)	O(1)	O(1)	O(1)	$O(V^2)$

Green - Constant
Orange - Linear
Red - Polynomial