CS2400 - Data Structures and Advanced Programming
Module 12: Graphs

Hao Ji

Computer Science Department

Cal Poly Pomona

Today

- This Class
  - Graph Terminology
  - Graph Representations
  - Graph Traversals

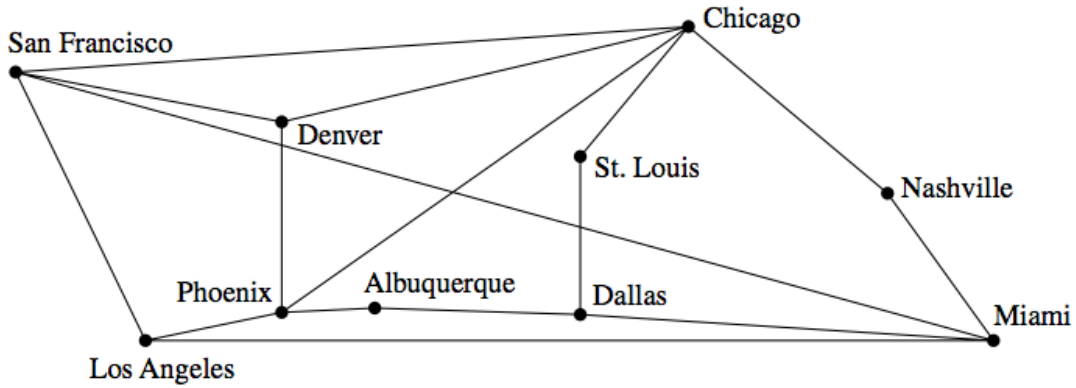
Graph Terminology

- Graphs are the most general data structure. They are also commonly used data structures.

- Graph is a non-linear data structure consisting of nodes and edges between nodes.

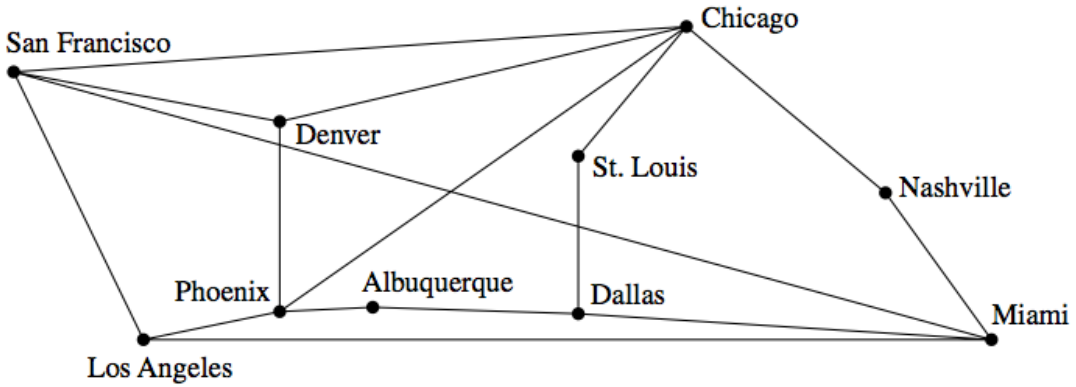
$$G = (V, E)$$

where V is a set of vertices (nodes) and E is a set of edges



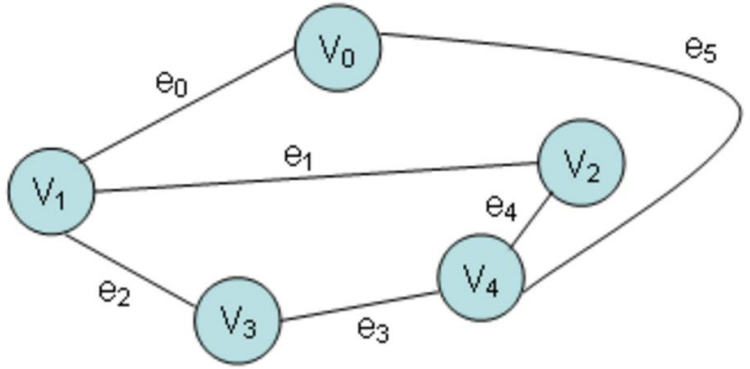
Graph Terminology

- Example
  - The set of nodes in the airline map below is **{Chicago, Nashville, Miami, Dallas, St. Louis, Albuquerque, Phoenix, Denver, San Francisco, Los Angeles}.**
  - There are 16 arcs, including **Phoenix–Albuquerque, Chicago–Nashville,** and **Miami–Dallas**.



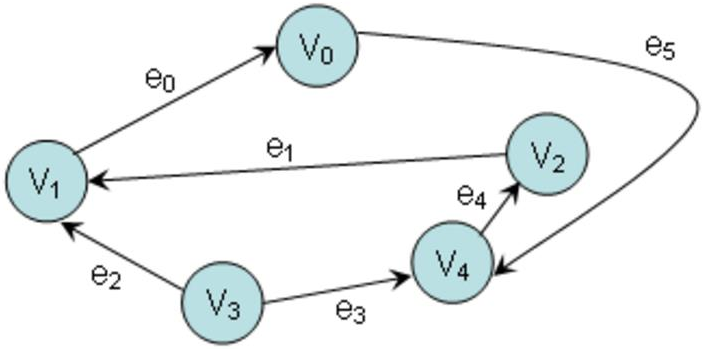
Graph Terminology

- Types of graphs
  - Undirected graph



Undirected graph

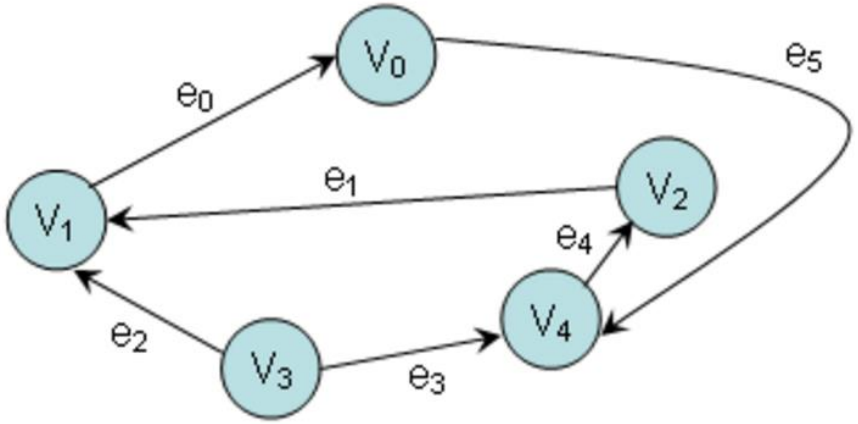
  - Directed graph



Directed graph

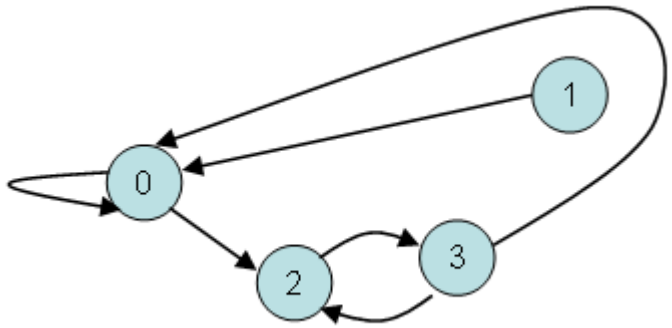
Graph Terminology

- Directed graph
  - Each edge is associated with two vertices, called its **source** and **target** vertices.
  - The **order** of the two connected vertices is important.



Graph Terminology

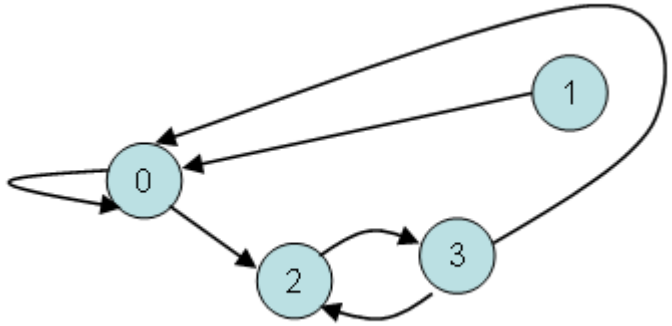
- **Loop**: an edge that connects a vertex to itself.
- **Path**: a sequence of vertices, p_0, p_1, ..., p_m, such that each adjacent pair of vertices p_i and p_{i+1} are connected by an edge.



Graph Terminology

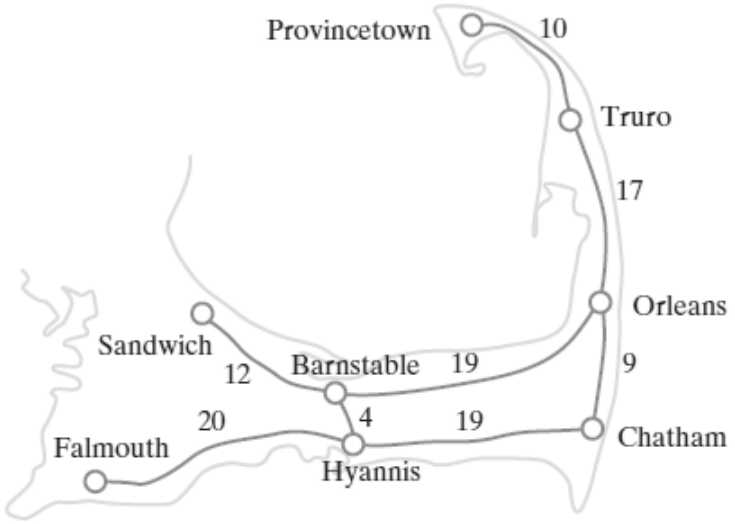
- **Loop**: an edge that connects a vertex to itself.
- **Path**: a sequence of vertices, p_0, p_1, ..., p_m, such that each adjacent pair of vertices p_i and p_{i+1} are connected by an edge.
- **Cycle**: a simple path with no repeated vertices or edges other than the starting and ending vertices. A cycle in a directed graph is called a directed cycle.

In fact, a tree is a special type of graph without cycle



Graph Terminology

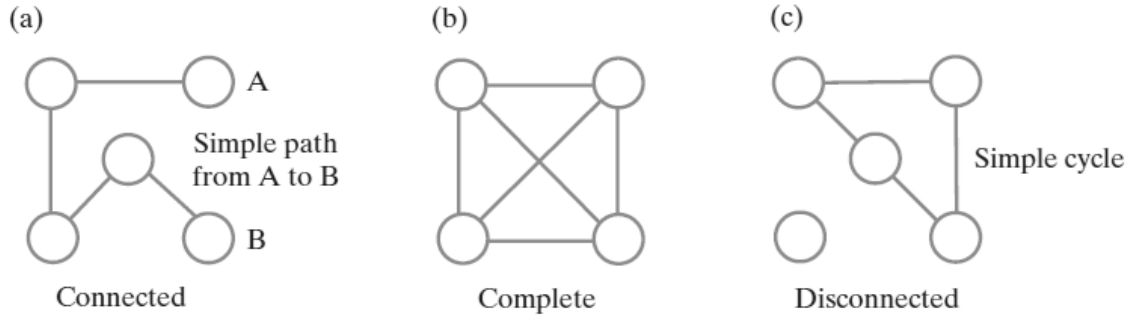
- A **weighted** graph, has values on its edges
  - Values are called either weights or costs



A weighted graph

Graph Terminology

- Connected graph
  - A graph that has a path between every pair of distinct vertices.
- Complete graph
  - A graph that has an edge between every pair of distinct vertices.
- Undirected graphs can be
  - Connected
  - Complete or
  - Disconnected



Graph Terminology

- **Adjacent** Vertices
  - In an undirected graph, two vertices are **adjacent** if they are joined by an edge.
  - In a directed graph, vertex *A* is **adjacent** to vertex *B* if a directed edge begins at *B* and ends at *A*.

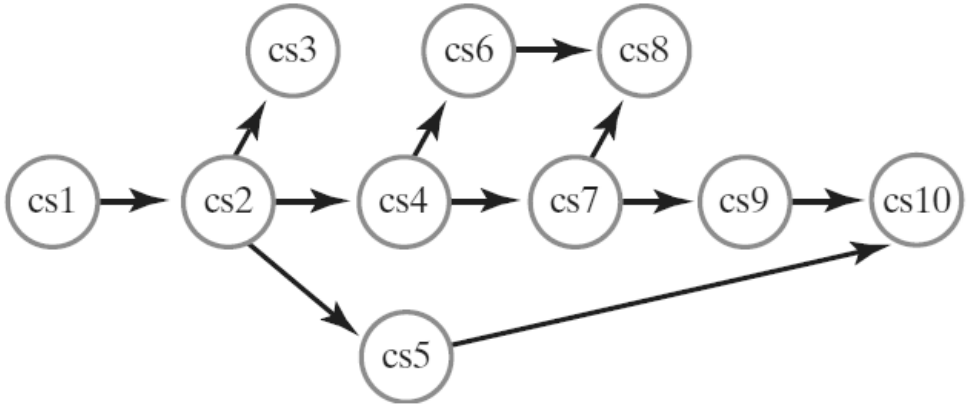
- Adjacent vertices are called **neighbors**.

Vertex A and vertex B are adjacent

Vertex A is adjacent to vertex B, but B is not adjacent to A

In-Class Exercise

- A graph about Course Prerequisites
  - Why is it directed?
  - Is cs1 adjacent to cs2?
  - Is cs1 adjacent to cs4?
  - Is cs2 adjacent to cs1?
  - Is cs4 adjacent to cs1?



In-Class Exercise

- Number of Edges in a Graph

- If an undirected graph has n vertices,
  - What is the maximum number of edges that the undirected graph can have?

- If a directed graph has n vertices,
  - What is the maximum number of edges that the directed graph can have?

In-Class Exercise

- Number of Edges in a Graph

- If an undirected graph has n vertices,
  - What is the maximum number of edges that the undirected graph can have?

    n (n - 1) / 2 edges if the graph is undirected
- If a directed graph has n vertices,
  - What is the maximum number of edges that the directed graph can have?

In-Class Exercise

- Number of Edges in a Graph

- If an undirected graph has n vertices,
  - What is the maximum number of edges that the undirected graph can have?

***n* (*n* - 1) / 2** edges if the graph is undirected

- If a directed graph has n vertices,
  - What is the maximum number of edges that the directed graph can have?

n (n - 1) *edges if the graph is directed*

Graph Implementations

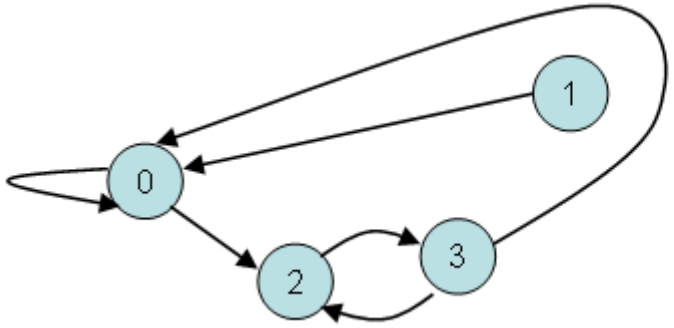
- Two common implementations of the ADT graph
  - Use either an array or a list

- The array is typically a two-dimensional array called an adjacency matrix

- The list is called an adjacency list.

An Adjacency Matrix

- Representing Graphs with an Adjacency Matrix
  - An adjacency matrix is a square grid of true/false values that represent the edges of a graph.
  - If the graph contains *n* vertices, then the grid contains *n* rows and *n* columns.
  - For two vertex numbers *i* and *j*, the component at row *i* and column *j* is
    - True, if there is an edge from vertex *i* to vertex *j*;
    - otherwise, the component is false.



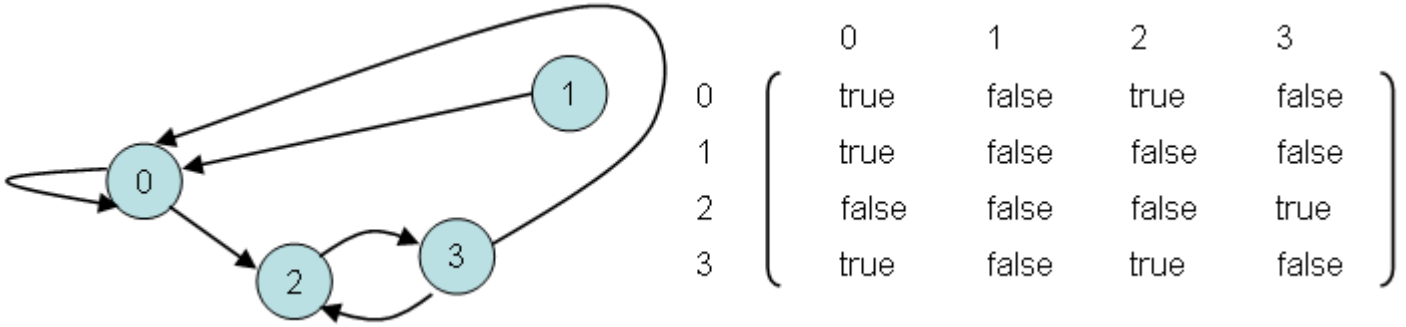
	0	1	2	3
0	true	false	true	false
1	true	false	false	false
2	false	false	false	true
3	true	false	true	false

An Adjacency Matrix

- We can use a two-dimensional array to store an adjacency matrix:

boolean[][] adjacent = new boolean[4][4];

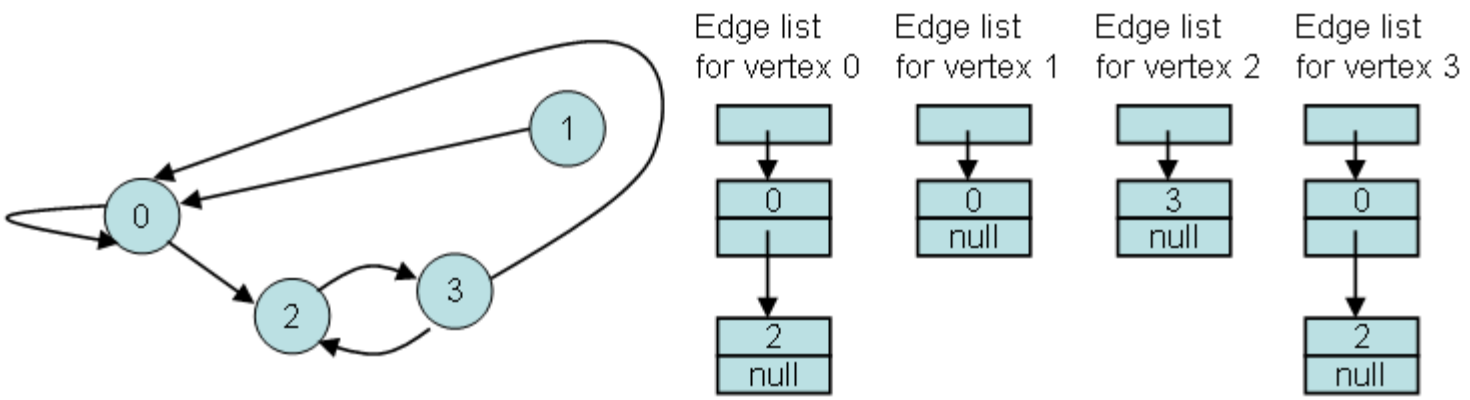
- Once the adjacency matrix has been set, an application can examine locations of the matrix to determine which edges are present and which are missing.



	0	1	2	3
0	true	false	true	false
1	true	false	false	false
2	false	false	false	true
3	true	false	true	false

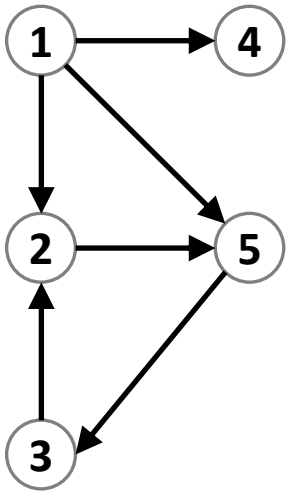
An Adjacency List

- Representing Graphs with Adjacency Lists
  - A directed graph with *n* vertices can be represented by *n* different linked lists.
  - List number *i* provides the connections for vertex *i*.
  - For each entry *j* in list number *i*, there is an edge from *i* to *j*.



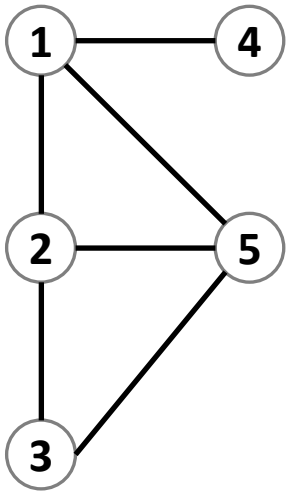
In-Class Exercises

- Use adjacency matrix and adjacency list to represent the following graphs?



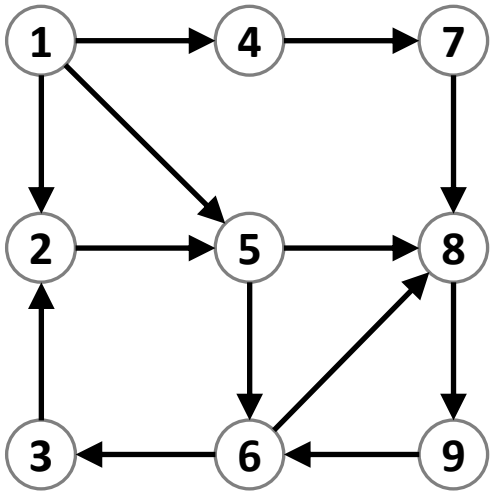
In-Class Exercises

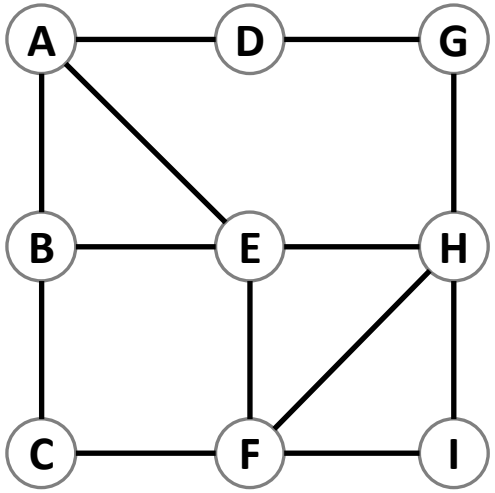
- Use adjacency matrix and adjacency list to represent the following graphs?



In-Class Exercises

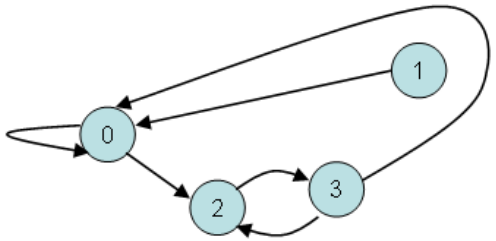
- Use adjacency matrix and adjacency list to represent the following graphs?

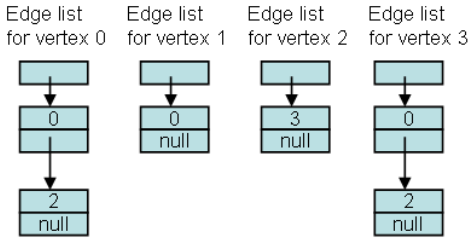




Which Representation is Better?

	Adjacency list	Adjacency matrix
Store graph	$O(\|V\| + \|E\|)$	$O(\|V\|^2)$
Add vertex	$O(1)$	$O(\|V\|^2)$
Add edge	$O(1)$	$O(1)$
Remove vertex	$O(\|E\|)$	$O(\|V\|^2)$
Remove edge	$O(\|V\|)$	$O(1)$
Query: are vertices *x* and *y* adjacent? (assuming that their storage positions are known)	$O(\|V\|)$	$O(1)$
Remarks	Slow to remove vertices and edges, because it needs to find all vertices or edges	Slow to add or remove vertices, because matrix must be resized/copied





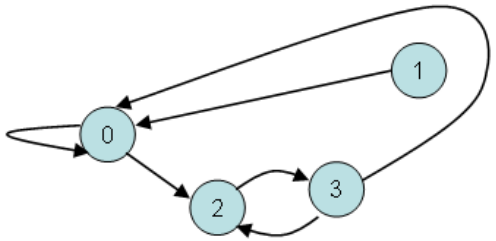
	0	1	2	3
0	true	false	true	false
1	true	false	false	false
2	false	false	false	true
3	true	false	true	false

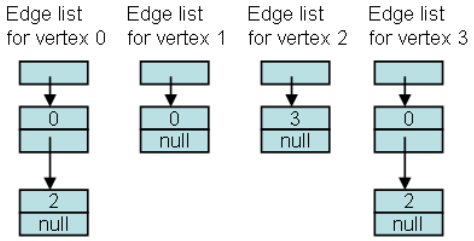
https://en.wikipedia.org/wiki/Graph_(abstract_data_type)

If the space is available, **an adjacency matrix** is easier to implement and is generally easier to use than adjacency list.

Which Representation is Better?

	Adjacency list	Adjacency matrix
Store graph	$O(\|V\| + \|E\|)$	$O(\|V\|^2)$
Add vertex	$O(1)$	$O(\|V\|^2)$
Add edge	$O(1)$	$O(1)$
Remove vertex	$O(\|E\|)$	$O(\|V\|^2)$
Remove edge	$O(\|V\|)$	$O(1)$
Query: are vertices *x* and *y* adjacent? (assuming that their storage positions are known)	$O(\|V\|)$	$O(1)$
Remarks	Slow to remove vertices and edges, because it needs to find all vertices or edges	Slow to add or remove vertices, because matrix must be resized/copied



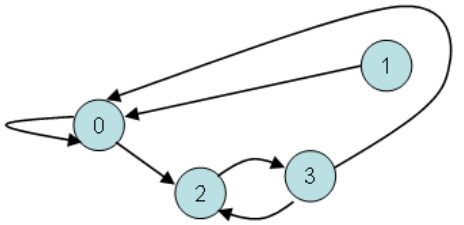


	0	1	2	3
0	true	false	true	false
1	true	false	false	false
2	false	false	false	true
3	true	false	true	false

https://en.wikipedia.org/wiki/Graph_(abstract_data_type)

Graph Implementations

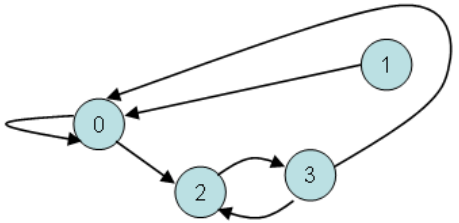
```
public class Graph<E>
{
    private boolean[][] edges; // edges[i][j] is true if there is a vertex from i to j
    private E[] labels; // labels[i] contains the label for vertex i

  ...
  // Constructors
  // Boolean method
  //     isEdge(int source, int target)
  // Methods:
  //      addEdge(int source, int target) .
  //      getLabel(int vertex).
  //      neighbors(int vertex)
  //      removeEdge(int source, int target)
  //      setLabel(int vertex, E newLabel)
  //      size()
}
```



	0	1	2	3
0	true	false	true	false
1	true	false	false	false
2	false	false	false	true
3	true	false	true	false

Graph Implementations

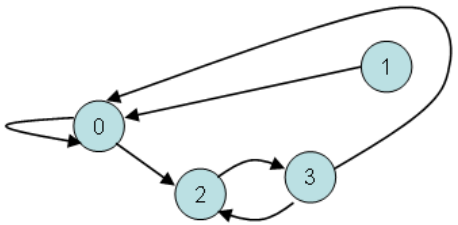
```
public class Graph<E>

{
    private boolean[][] edges; // edges[i][j] is true if there is a vertex from i to j
    private E[] labels; // labels[i] contains the label for vertex i


    public Graph(int n) {
        edges = new boolean[n][n]; // All values initially false
        labels = (E[]) new Object[n]; // All values initially null
    }

   ...

}
```



	0	1	2	3
0	true	false	true	false
1	true	false	false	false
2	false	false	false	true
3	true	false	true	false

Graph Implementations

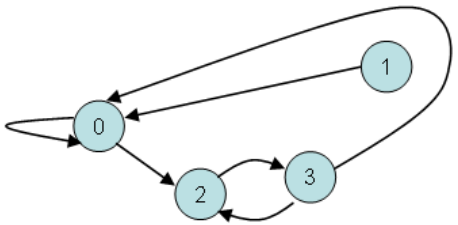
```
// Accessor method to get the label of a vertex of this Graph
public E getLabel(int vertex) {
    return labels[vertex];
}


// Test whether an edge exists
public boolean isEdge(int source, int target) {
    return edges[source][target];
}


// Add an edge
public void addEdge(int source, int target) {
    edges[source][target] = true;
}
```



	0	1	2	3
0	true	false	true	false
1	true	false	false	false
2	false	false	false	true
3	true	false	true	false

Graph Implementations

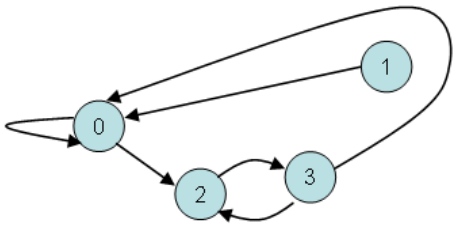
```
// Obtain a list of neighbors of a specified vertex of this Graph
public int[] neighbors(int vertex) {
    int i;
    int count = 0;
    int[] answer;

    for (i = 0; i < labels.length; i++) {
        if (edges[vertex][i])
            count++;
    }
    answer = new int[count];
    count = 0;
    for (i = 0; i < labels.length; i++) {
        if (edges[vertex][i])
            answer[count++] = i;
    }
    return answer;
}
```



	0	1	2	3
0	true	false	true	false
1	true	false	false	false
2	false	false	false	true
3	true	false	true	false

Graph Implementations

```
// Remove an edge
public void removeEdge(int source, int target) {
    edges[source][target] = false;
}

// Change the label of a vertex of this Graph
public void setLabel(int vertex, E newLabel) {
    labels[vertex] = newLabel;
}

// Accessor method to determine the number of vertices in this Graph
public int size() {
    return labels.length;
}
```

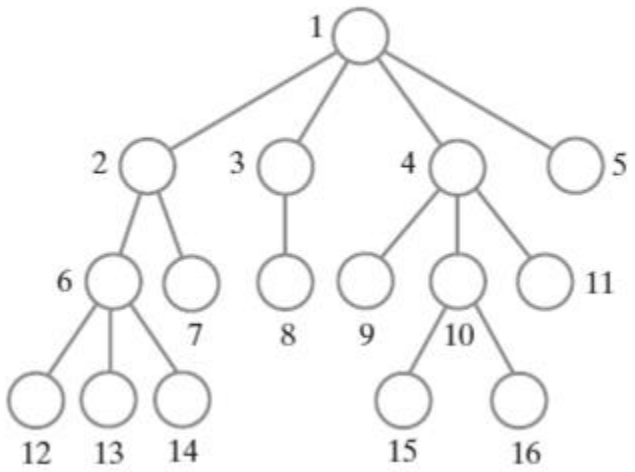


	0	1	2	3
0	true	false	true	false
1	true	false	false	false
2	false	false	false	true
3	true	false	true	false

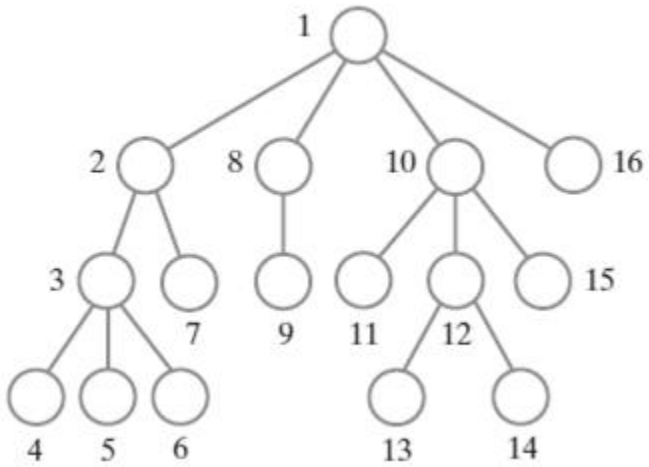
Graph Traversals

- Breadth-first search
- Depth-first search

Recall: Tree Traversals



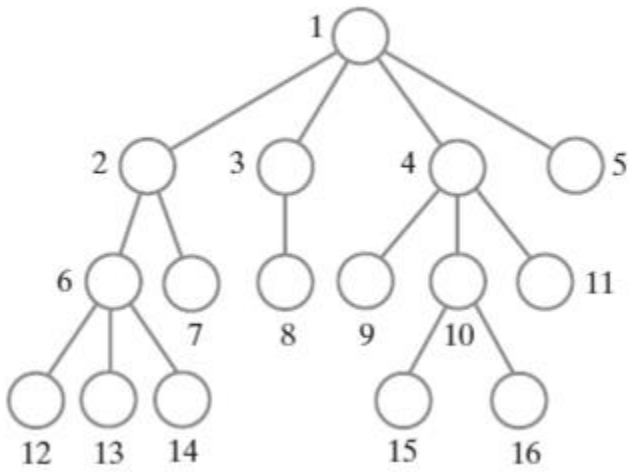
Level-order



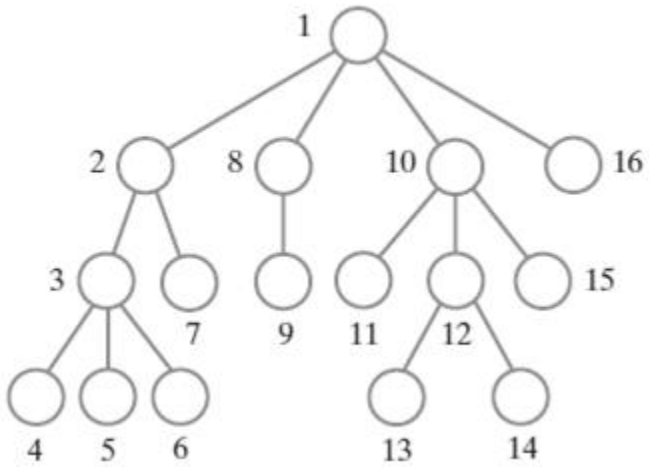
Pre-order

Graph Traversals

- When we see it as a **Graph**



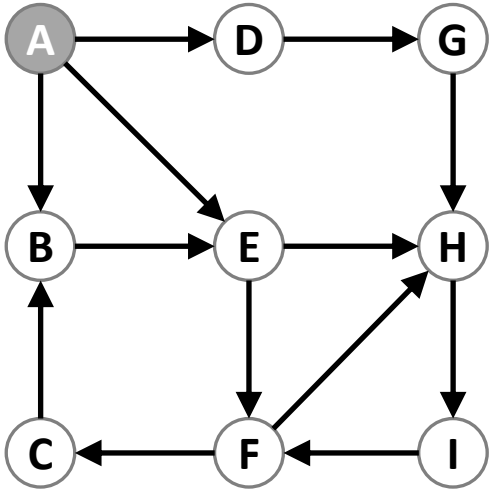
Breadth-first traversal



Depth-first traversal

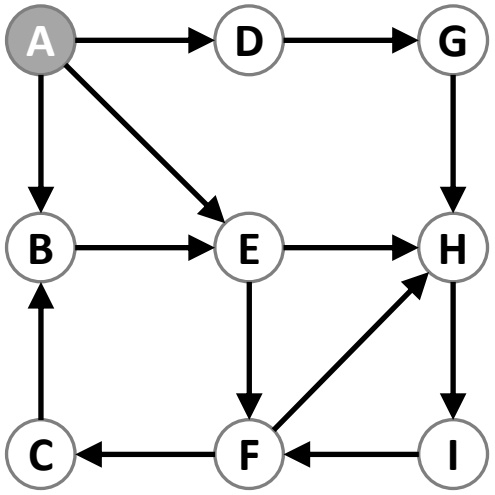
Graph Traversals

- Consider a general graph
  - **Origin vertex**: a vertex that a graph traversal starts from



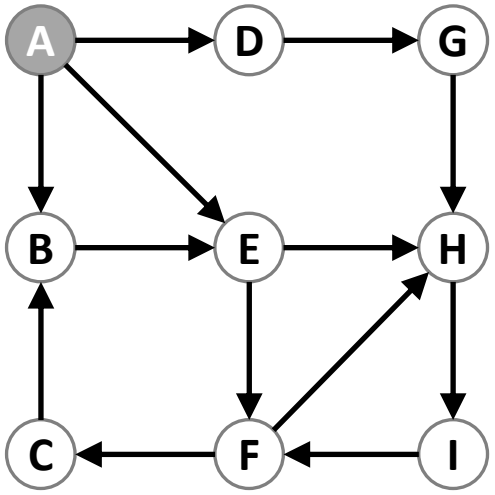
Breadth-First Traversal

- Given an origin vertex, a breadth-first traversal
  - **Visits the origin and the origin's neighbors.**
  - Then, considers each of these neighbors and visits their neighbors.



Breadth-First Traversal

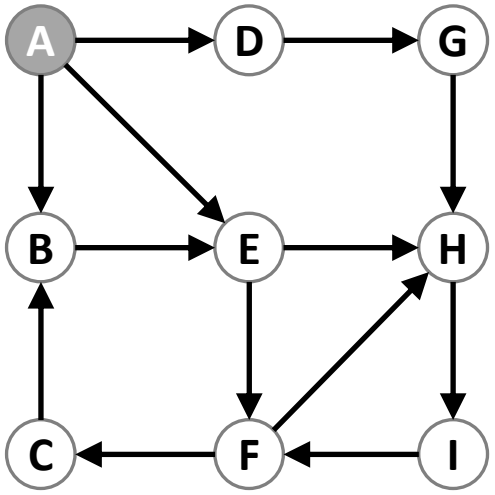
- Given an origin vertex, a breadth-first traversal
  - **Visits the origin and the origin's neighbors.**
  - Then, considers each of these neighbors and visits their neighbors.



- The traversal uses a queue to hold the visited vertices.

Breadth-First Traversal

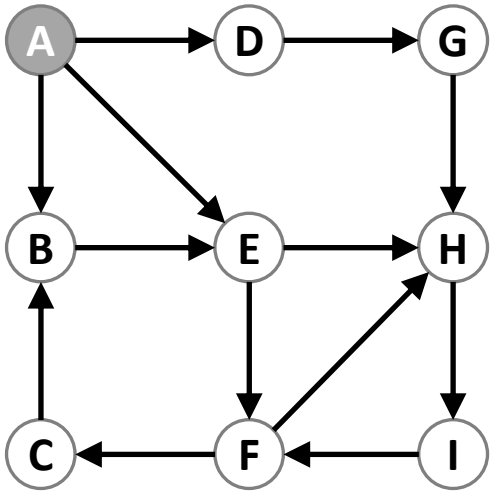
- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - **Then, considers each of these neighbors and visits their neighbors.**



- The traversal uses a queue to hold the visited vertices.

Breadth-First Traversal

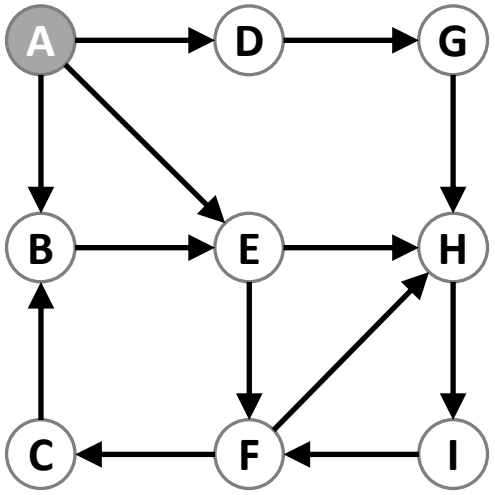
- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - Then, considers each of these neighbors and visits their neighbors.



frontVertex	nextNeighbor	Visited vertex	vertexQueue (front to back)	traversalOrder (front to back)
		A	A	A
A			*empty*	
	B	B	B	AB
	D	D	BD	ABD
	E	E	BDE	ABDE
B			DE	
D			E	
	G	G	EG	ABDEG
E			G	
	F	F	GF	ABDEGF
	H	H	GFH	ABDEGFH
G			FH	
F			H	
	C	C	HC	ABDEGFHC
H			C	
	I	I	CI	ABDEGFHCI
C			I	
I			*empty*	

Breadth-First Traversal

- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - Then, considers each of these neighbors and visits their neighbors.

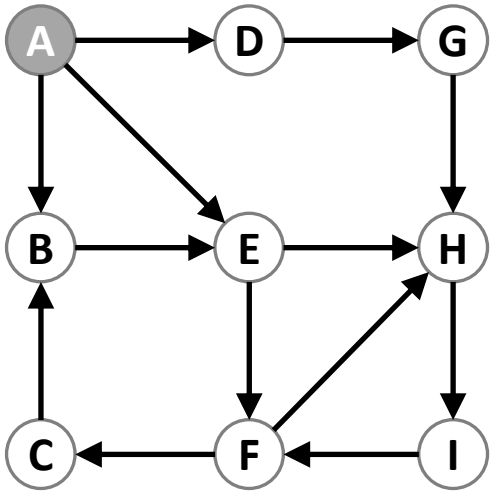


frontVertex	nextNeighbor	Visited vertex	vertexQueue (front to back)	traversalOrder (front to back)

- The traversal order is the order in which vertices are added to the queue. (Note: we can retain this traversal order in a second queue.)

Breadth-First Traversal

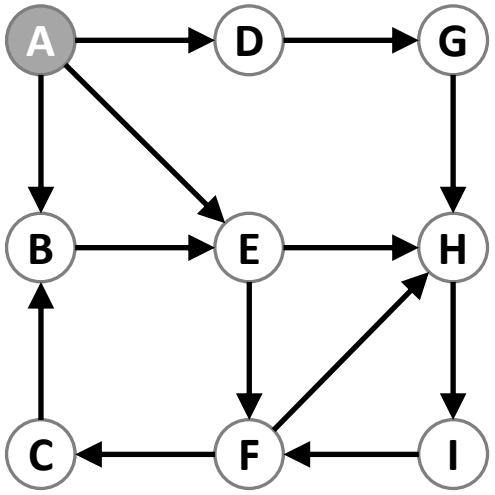
- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - Then, considers each of these neighbors and visits their neighbors.



frontVertex	nextNeighbor	Visited vertex	vertexQueue (front to back)	traversalOrder (front to back)
		A		

Breadth-First Traversal

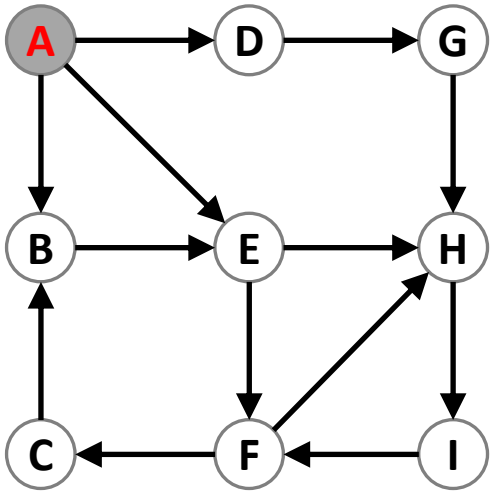
- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - Then, considers each of these neighbors and visits their neighbors.



frontVertex	nextNeighbor	Visited vertex	vertexQueue (front to back)	traversalOrder (front to back)
		A	A	A

Breadth-First Traversal

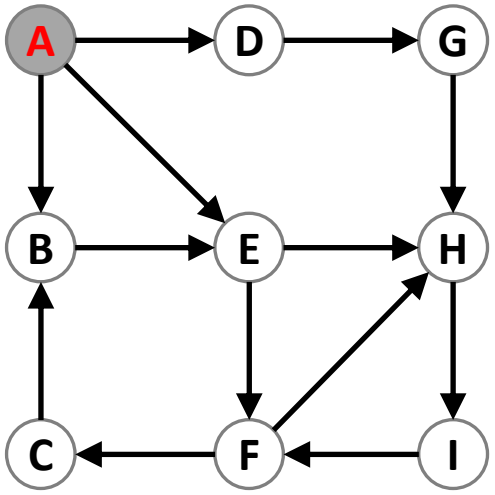
- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - Then, considers each of these neighbors and visits their neighbors.



frontVertex	nextNeighbor	Visited vertex	vertexQueue (front to back)	traversalOrder (front to back)
		A	A	A
A			*empty*	

Breadth-First Traversal

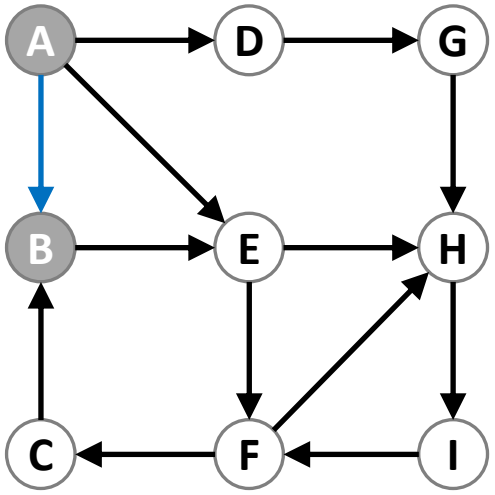
- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - Then, considers each of these neighbors and visits their neighbors.



frontVertex	nextNeighbor	Visited vertex	vertexQueue (front to back)	traversalOrder (front to back)
		A	A	A
A			*empty*	
	B			
	D			
	E			

Breadth-First Traversal

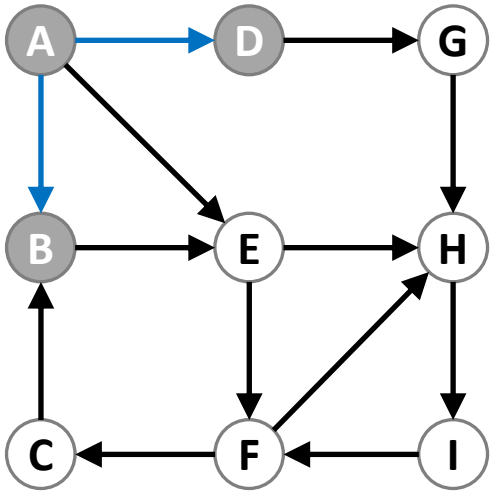
- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - Then, considers each of these neighbors and visits their neighbors.



frontVertex	nextNeighbor	Visited vertex	vertexQueue (front to back)	traversalOrder (front to back)
		A	A	A
A			*empty*	
	B	B	B	AB
	D			
	E			

Breadth-First Traversal

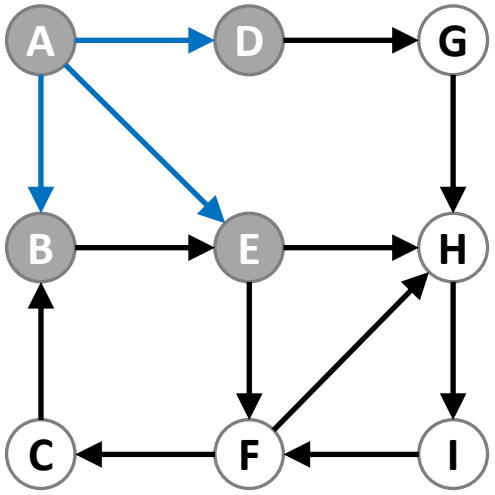
- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - Then, considers each of these neighbors and visits their neighbors.



frontVertex	nextNeighbor	Visited vertex	vertexQueue (front to back)	traversalOrder (front to back)
		A	A	A
A			*empty*	
	B	B	B	AB
	D	D	BD	ABD
	E			

Breadth-First Traversal

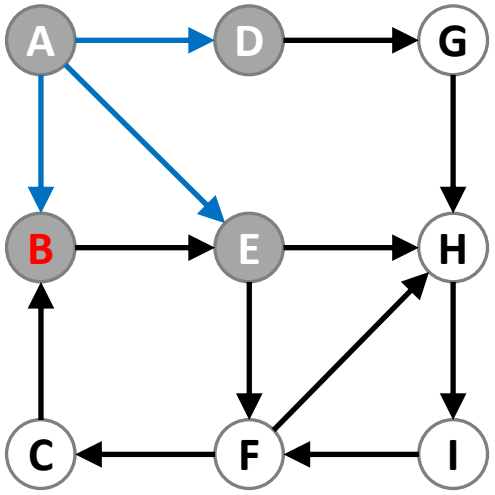
- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - Then, considers each of these neighbors and visits their neighbors.



frontVertex	nextNeighbor	Visited vertex	vertexQueue (front to back)	traversalOrder (front to back)
		A	A	A
A			*empty*	
	B	B	B	AB
	D	D	BD	ABD
	E	E	BDE	ABDE

Breadth-First Traversal

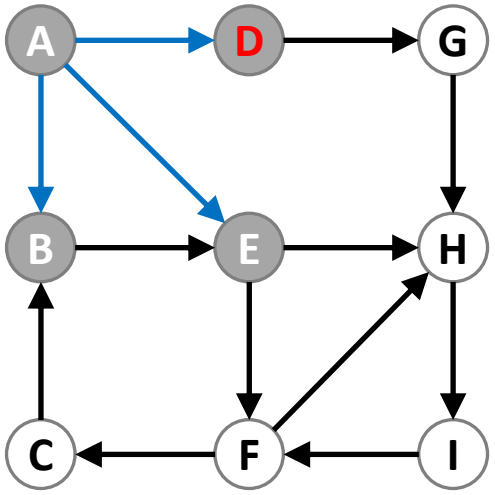
- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - Then, considers each of these neighbors and visits their neighbors.



frontVertex	nextNeighbor	Visited vertex	vertexQueue (front to back)	traversalOrder (front to back)
		A	A	A
A			*empty*	
	B	B	B	AB
	D	D	BD	ABD
	E	E	BDE	ABDE
B			DE	

Breadth-First Traversal

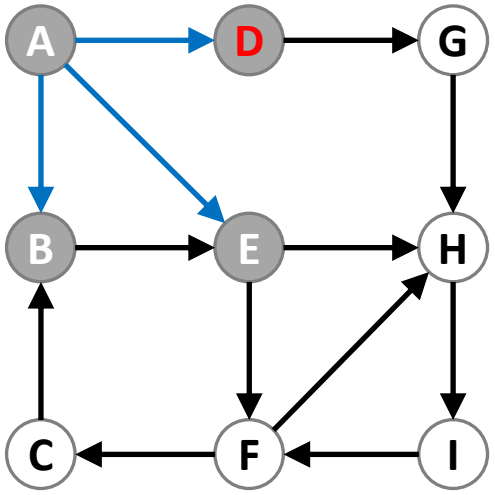
- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - Then, considers each of these neighbors and visits their neighbors.



frontVertex	nextNeighbor	Visited vertex	vertexQueue (front to back)	traversalOrder (front to back)
		A	A	A
A			*empty*	
	B	B	B	AB
	D	D	BD	ABD
	E	E	BDE	ABDE
B			DE	
D			E	

Breadth-First Traversal

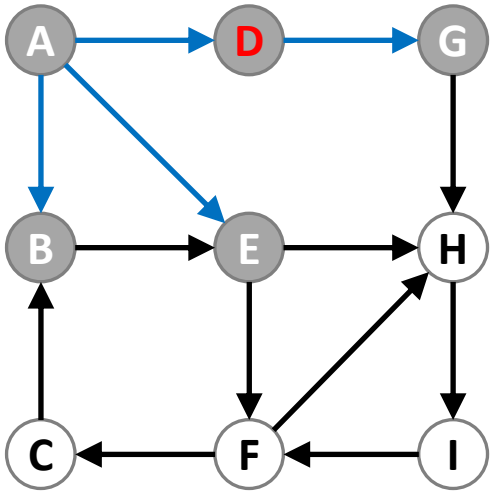
- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - Then, considers each of these neighbors and visits their neighbors.



frontVertex	nextNeighbor	Visited vertex	vertexQueue (front to back)	traversalOrder (front to back)
		A	A	A
A			*empty*	
	B	B	B	AB
	D	D	BD	ABD
	E	E	BDE	ABDE
B			DE	
D			E	
	G			

Breadth-First Traversal

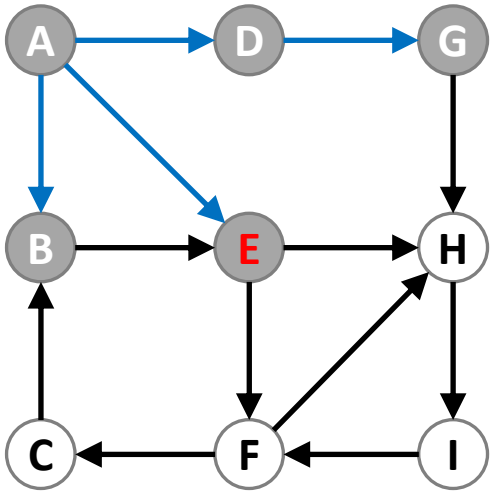
- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - Then, considers each of these neighbors and visits their neighbors.



frontVertex	nextNeighbor	Visited vertex	vertexQueue (front to back)	traversalOrder (front to back)
		A	A	A
A			*empty*	
	B	B	B	AB
	D	D	BD	ABD
	E	E	BDE	ABDE
B			DE	
D			E	
	G	G	EG	ABDEG

Breadth-First Traversal

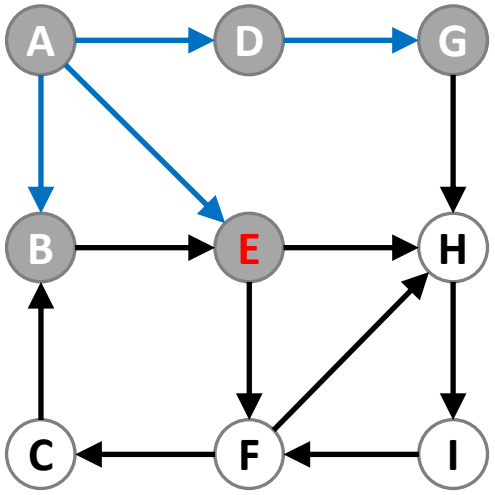
- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - Then, considers each of these neighbors and visits their neighbors.



frontVertex	nextNeighbor	Visited vertex	vertexQueue (front to back)	traversalOrder (front to back)
		A	A	A
A			*empty*	
	B	B	B	AB
	D	D	BD	ABD
	E	E	BDE	ABDE
B			DE	
D			E	
	G	G	EG	ABDEG
E			G	

Breadth-First Traversal

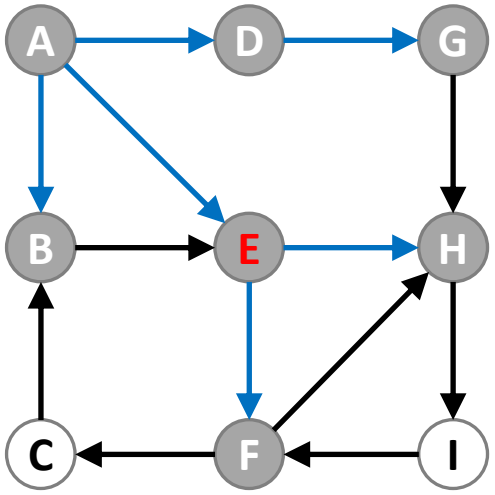
- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - Then, considers each of these neighbors and visits their neighbors.



frontVertex	nextNeighbor	Visited vertex	vertexQueue (front to back)	traversalOrder (front to back)
		A	A	A
A			*empty*	
	B	B	B	AB
	D	D	BD	ABD
	E	E	BDE	ABDE
B			DE	
D			E	
	G	G	EG	ABDEG
E			G	
	F			
	H			

Breadth-First Traversal

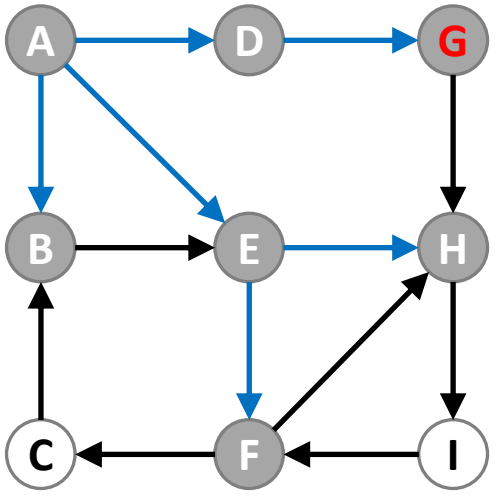
- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - Then, considers each of these neighbors and visits their neighbors.



frontVertex	nextNeighbor	Visited vertex	vertexQueue (front to back)	traversalOrder (front to back)
		A	A	A
A			*empty*	
	B	B	B	AB
	D	D	BD	ABD
	E	E	BDE	ABDE
B			DE	
D			E	
	G	G	EG	ABDEG
E			G	
	F	F	GF	ABDEGF
	H	H	GFH	ABDEGFH

Breadth-First Traversal

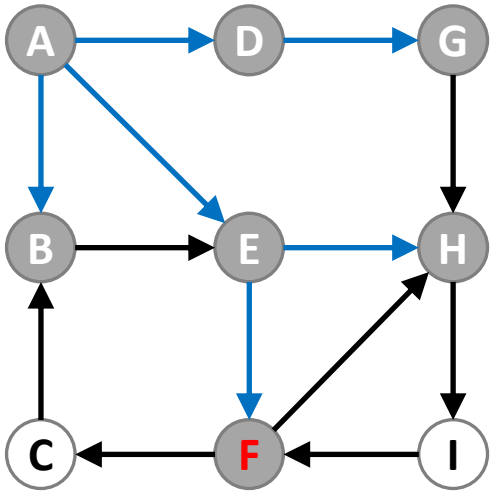
- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - Then, considers each of these neighbors and visits their neighbors.



frontVertex	nextNeighbor	Visited vertex	vertexQueue (front to back)	traversalOrder (front to back)
		A	A	A
A			*empty*	
	B	B	B	AB
	D	D	BD	ABD
	E	E	BDE	ABDE
B			DE	
D			E	
	G	G	EG	ABDEG
E			G	
	F	F	GF	ABDEGF
	H	H	GFH	ABDEGFH
G			FH	

Breadth-First Traversal

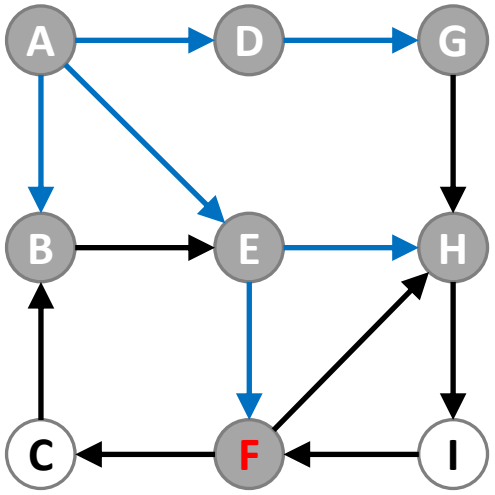
- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - Then, considers each of these neighbors and visits their neighbors.



frontVertex	nextNeighbor	Visited vertex	vertexQueue (front to back)	traversalOrder (front to back)
		A	A	A
A			*empty*	
	B	B	B	AB
	D	D	BD	ABD
	E	E	BDE	ABDE
B			DE	
D			E	
	G	G	EG	ABDEG
E			G	
	F	F	GF	ABDEGF
	H	H	GFH	ABDEGFH
G			FH	
F			H	

Breadth-First Traversal

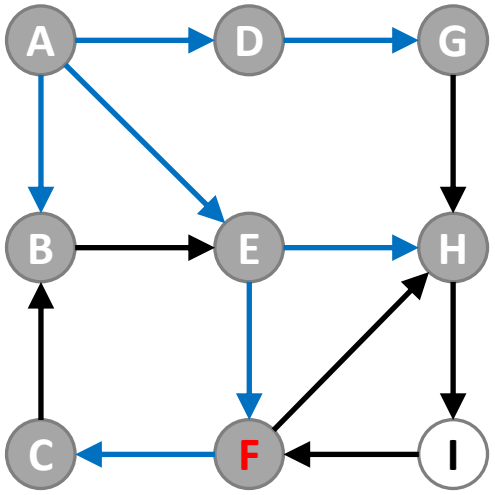
- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - Then, considers each of these neighbors and visits their neighbors.



frontVertex	nextNeighbor	Visited vertex	vertexQueue (front to back)	traversalOrder (front to back)
		A	A	A
A			*empty*	
	B	B	B	AB
	D	D	BD	ABD
	E	E	BDE	ABDE
B			DE	
D			E	
	G	G	EG	ABDEG
E			G	
	F	F	GF	ABDEGF
	H	H	GFH	ABDEGFH
G			FH	
F			H	
	C			

Breadth-First Traversal

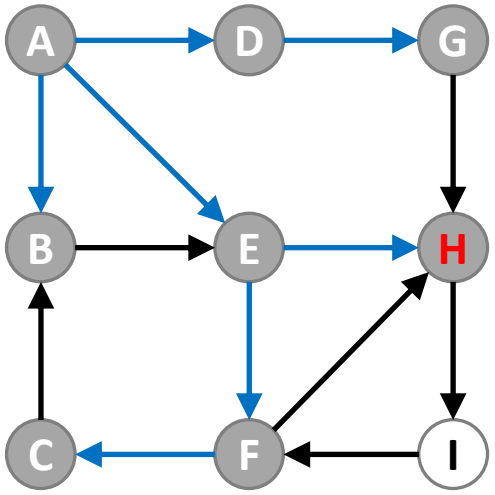
- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - Then, considers each of these neighbors and visits their neighbors.



frontVertex	nextNeighbor	Visited vertex	vertexQueue (front to back)	traversalOrder (front to back)
		A	A	A
A			*empty*	
	B	B	B	AB
	D	D	BD	ABD
	E	E	BDE	ABDE
B			DE	
D			E	
	G	G	EG	ABDEG
E			G	
	F	F	GF	ABDEGF
	H	H	GFH	ABDEGFH
G			FH	
F			H	
	C	C	HC	ABDEGFHC

Breadth-First Traversal

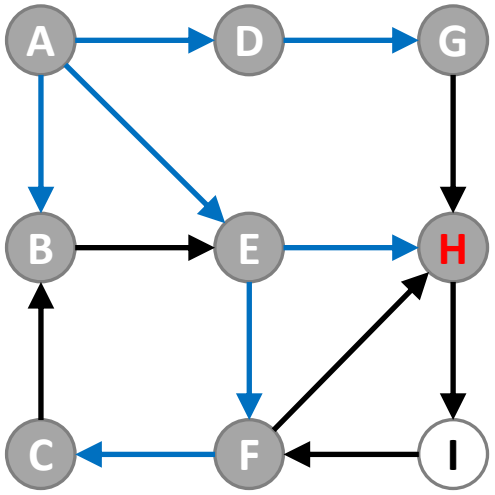
- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - Then, considers each of these neighbors and visits their neighbors.



frontVertex	nextNeighbor	Visited vertex	vertexQueue (front to back)	traversalOrder (front to back)
		A	A	A
A			*empty*	
	B	B	B	AB
	D	D	BD	ABD
	E	E	BDE	ABDE
B			DE	
D			E	
	G	G	EG	ABDEG
E			G	
	F	F	GF	ABDEGF
	H	H	GFH	ABDEGFH
G			FH	
F			H	
	C	C	HC	ABDEGFHC
H			C	

Breadth-First Traversal

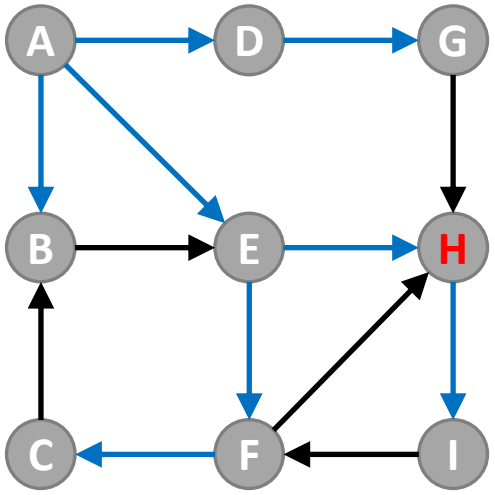
- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - Then, considers each of these neighbors and visits their neighbors.



frontVertex	nextNeighbor	Visited vertex	vertexQueue (front to back)	traversalOrder (front to back)
		A	A	A
A			*empty*	
	B	B	B	AB
	D	D	BD	ABD
	E	E	BDE	ABDE
B			DE	
D			E	
	G	G	EG	ABDEG
E			G	
	F	F	GF	ABDEGF
	H	H	GFH	ABDEGFH
G			FH	
F			H	
	C	C	HC	ABDEGFHC
H			C	
	I			

Breadth-First Traversal

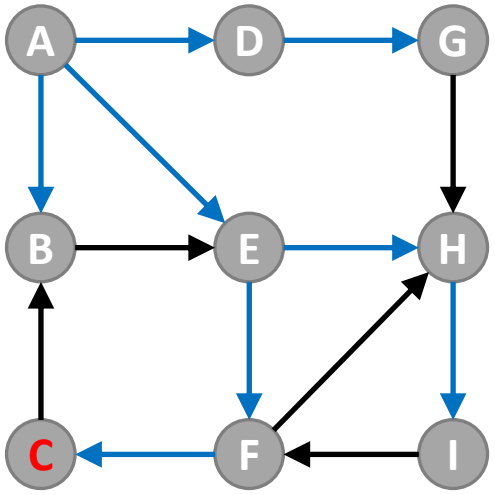
- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - Then, considers each of these neighbors and visits their neighbors.



frontVertex	nextNeighbor	Visited vertex	vertexQueue (front to back)	traversalOrder (front to back)
		A	A	A
A			*empty*	
	B	B	B	AB
	D	D	BD	ABD
	E	E	BDE	ABDE
B			DE	
D			E	
	G	G	EG	ABDEG
E			G	
	F	F	GF	ABDEGF
	H	H	GFH	ABDEGFH
G			FH	
F			H	
	C	C	HC	ABDEGFHC
H			C	
	I	I	CI	ABDEGFHCI

Breadth-First Traversal

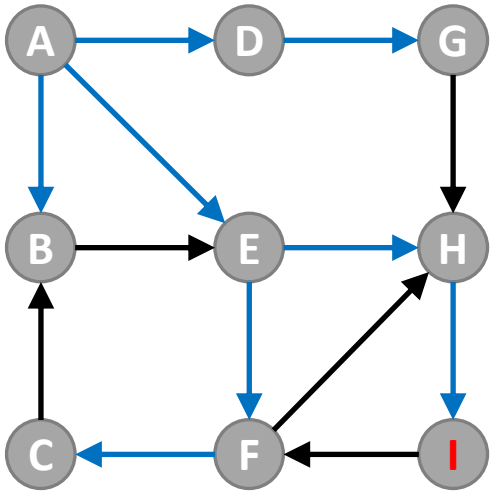
- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - Then, considers each of these neighbors and visits their neighbors.



frontVertex	nextNeighbor	Visited vertex	vertexQueue (front to back)	traversalOrder (front to back)
		A	A	A
A			*empty*	
	B	B	B	AB
	D	D	BD	ABD
	E	E	BDE	ABDE
B			DE	
D			E	
	G	G	EG	ABDEG
E			G	
	F	F	GF	ABDEGF
	H	H	GFH	ABDEGFH
G			FH	
F			H	
	C	C	HC	ABDEGFHC
H			C	
	I	I	CI	ABDEGFHCI
C				

Breadth-First Traversal

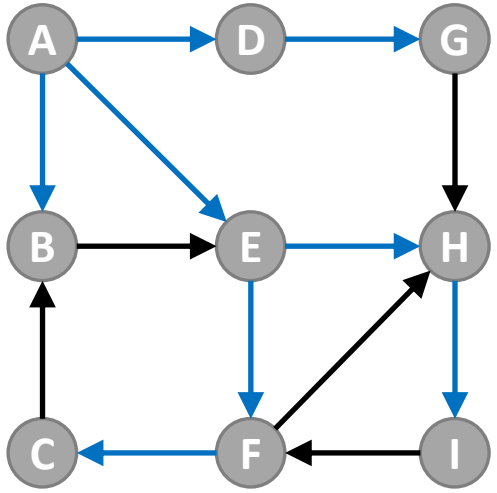
- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - Then, considers each of these neighbors and visits their neighbors.



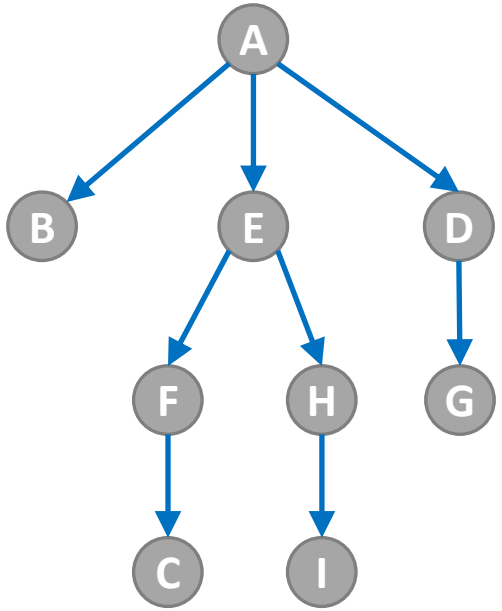
frontVertex	nextNeighbor	Visited vertex	vertexQueue (front to back)	traversalOrder (front to back)
		A	A	A
A			*empty*	
	B	B	B	AB
	D	D	BD	ABD
	E	E	BDE	ABDE
B			DE	
D			E	
	G	G	EG	ABDEG
E			G	
	F	F	GF	ABDEGF
	H	H	GFH	ABDEGFH
G			FH	
F			H	
	C	C	HC	ABDEGFHC
H			C	
	I	I	CI	ABDEGFHCI
C			I	
I			*empty*	

Breadth-First Traversal

- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - Then, considers each of these neighbors and visits their neighbors.



Breadth-First Traversal

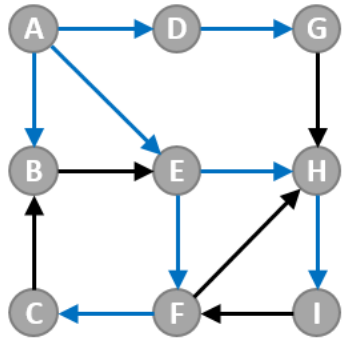


Paths form a breadth-first tree
(Order in which the nodes are expanded)

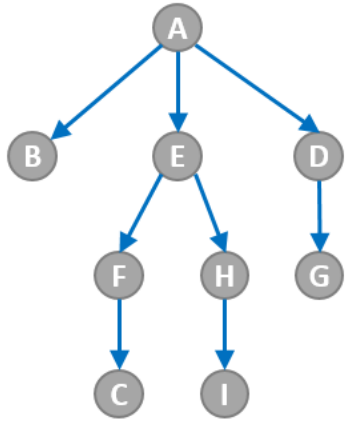
Breadth-First Traversal

- The time complexity can be expressed as O(|V|+|E|)
  - Every vertex and every edge will be explored in the worst case.

- In unweighted graphs, BFS gives the shortest path between two nodes u and v, where the path length is measured by number of edges.



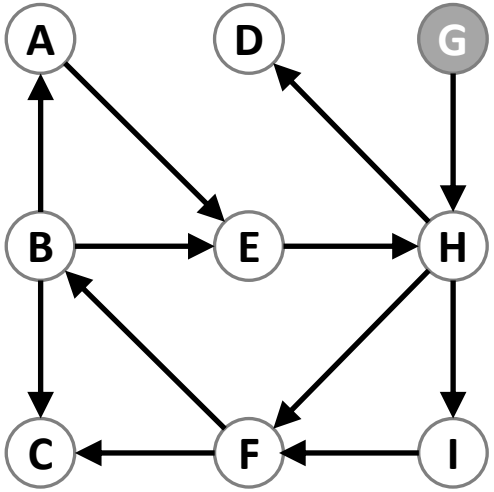
Breadth-First Traversal



Paths form a breadth-first tree
(Order in which the nodes are expanded)

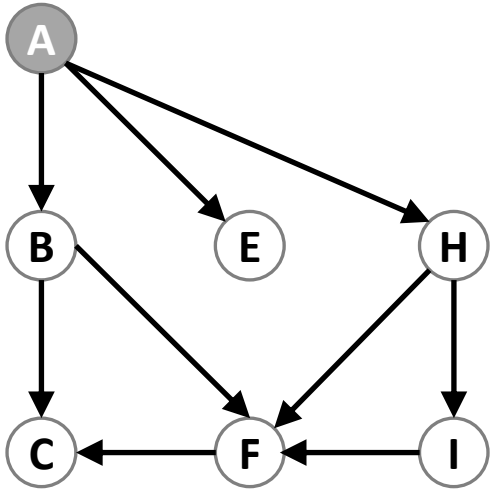
In-Class Exercise

- Perform breadth-first traversal beginning with Vertex *G*



In-Class Exercise

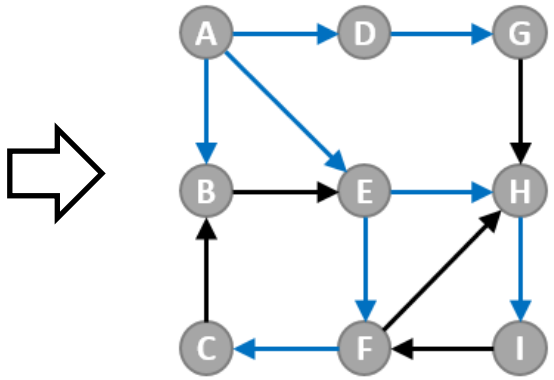
- Perform breadth-first traversal beginning with Vertex *A*



Implementation of BFS

- Breadth-first search uses a queue to keep track of vertices that still need to be visited.

```
Algorithm getBreadthFirstTraversal(originVertex)
traversalOrder = a new queue for the resulting traversal order
vertexQueue = a new queue to hold vertices as they are visited
Mark originVertex as visited
traversalOrder.enqueue(originVertex)
vertexQueue.enqueue(originVertex)

while (!vertexQueue.isEmpty())
{
   frontVertex = vertexQueue.dequeue()
   while (frontVertex has a neighbor)
   {
      nextNeighbor = next neighbor of frontVertex
      if (nextNeighbor is not visited)
      {
         Mark nextNeighbor as visited
         traversalOrder.enqueue(nextNeighbor)
         vertexQueue.enqueue(nextNeighbor)
      }
   }
}
return traversalOrder
```

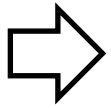


frontVertex	nextNeighbor	Visited vertex	vertexQueue (front to back)	traversalOrder (front to back)
		A	A	A
A			*empty*	
	B	B	B	AB
	D	D	BD	ABD
	E	E	BDE	ABDE
B			DE	
D			E	
	G	G	EG	ABDEG
E			G	
	F	F	GF	ABDEGF
	H	H	GFH	ABDEGFH
G			FH	
F			H	
	C	C	HC	ABDEGFHC
H			C	
	I	I	CI	ABDEGFHCI
C			I	
I			*empty*	

Implementation of BFS

- Breadth-first search uses a queue to keep track of vertices that still need to be visited.

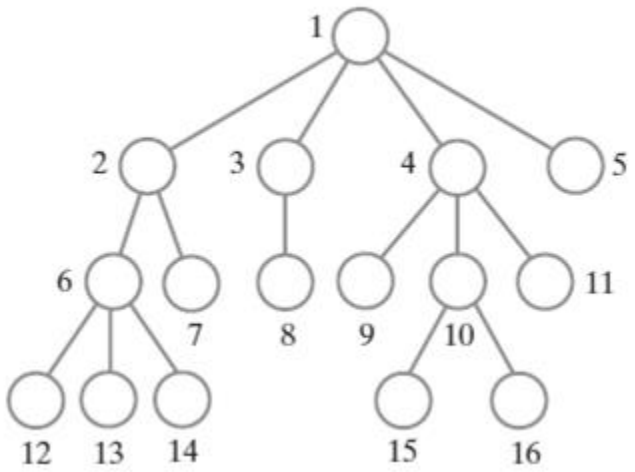
```
Algorithm getBreadthFirstTraversal(originVertex)
traversalOrder = a new queue for the resulting traversal order
vertexQueue = a new queue to hold vertices as they are visited
Mark originVertex as visited
traversalOrder.enqueue(originVertex)
vertexQueue.enqueue(originVertex)

while (!vertexQueue.isEmpty())
{
   frontVertex = vertexQueue.dequeue()
   while (frontVertex has a neighbor)
   {
      nextNeighbor = next neighbor of frontVertex
      if (nextNeighbor is not visited)
      {
         Mark nextNeighbor as visited
         traversalOrder.enqueue(nextNeighbor)
         vertexQueue.enqueue(nextNeighbor)
      }
   }
}
return traversalOrder
```



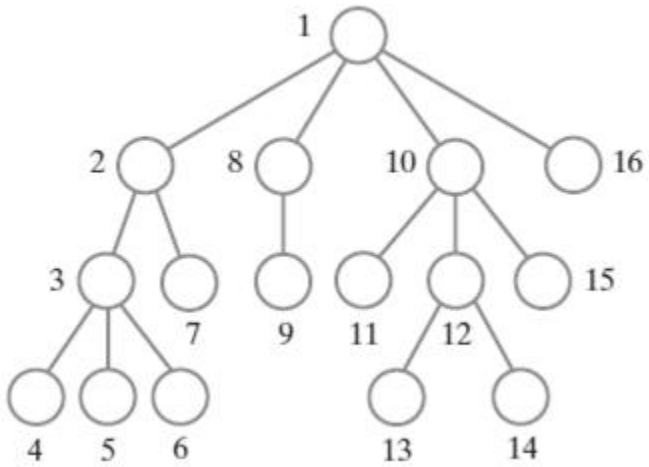
```
public QueueInterface<T> getBreadthFirstTraversal(T origin)
{
   resetVertices();
   QueueInterface<T> traversalOrder = new LinkedQueue<T>();
   QueueInterface<VertexInterface<T>> vertexQueue =
                                      new LinkedQueue<VertexInterface<T>>();

   VertexInterface<T> originVertex = vertices.getValue(origin);
   originVertex.visit();
   traversalOrder.enqueue(origin);    // enqueue vertex label
   vertexQueue.enqueue(originVertex); // enqueue vertex

   while (!vertexQueue.isEmpty())
   {
      VertexInterface<T> frontVertex = vertexQueue.dequeue();

      Iterator<VertexInterface<T>> neighbors =
                                   frontVertex.getNeighborIterator();
      while (neighbors.hasNext())
      {
         VertexInterface<T> nextNeighbor = neighbors.next();
         if (!nextNeighbor.isVisited())
         {
            nextNeighbor.visit();
            traversalOrder.enqueue(nextNeighbor.getLabel());
            vertexQueue.enqueue(nextNeighbor);
         } // end if
      } // end while
   } // end while

   return traversalOrder;
} // end getBreadthFirstTraversal
```

Graph Traversals

- When we see it as a graph



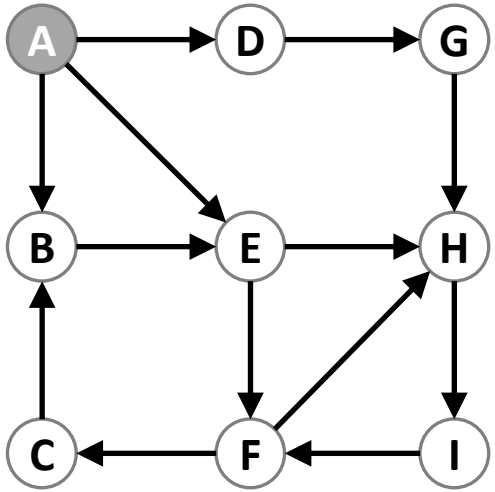
Breadth-first traversal



Depth-first traversal

Depth-First Traversal

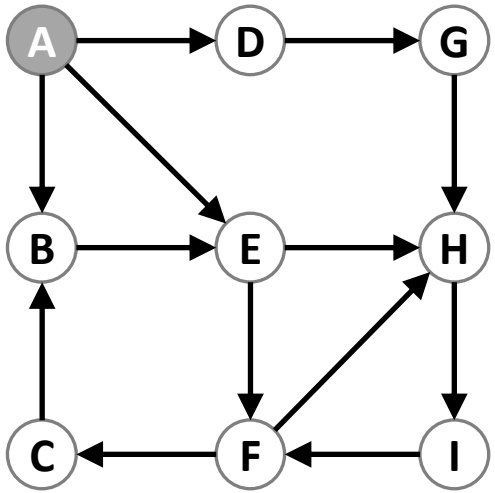
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backing up by one vertex, it considers another neighbor.



- The traversal uses a stack (can also implement it recursively) to expand the deepest unvisited nodes.

Depth-First Traversal

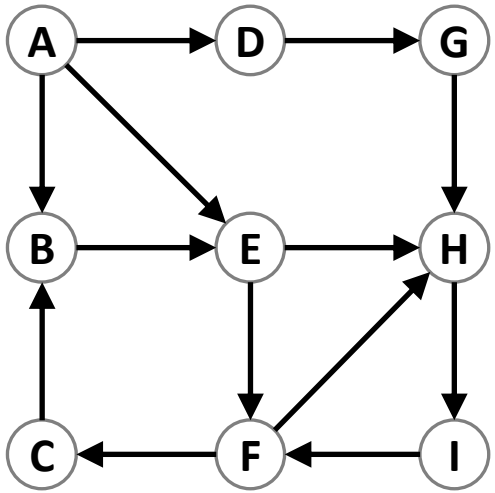
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backing up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	
F			FEBA	
	H	H	HFEBA	ABEFCH
H			HFEBA	
	I	I	IHFEBA	ABEFCHI
I			HFEBA	
H			FEBA	
F			EBA	
E			BA	
B			A	
A			A	
	D	D	DA	ABEFCHID
D			DA	
	G		GDA	ABEFCHIDG
G			DA	
D			A	
A			*empty*	ABEFCHIDG

Depth-First Traversal

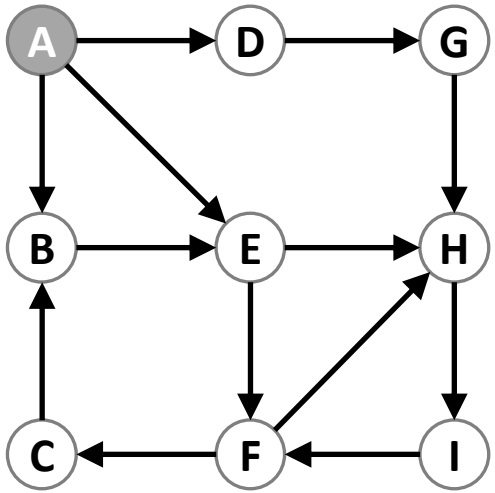
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)

Depth-First Traversal

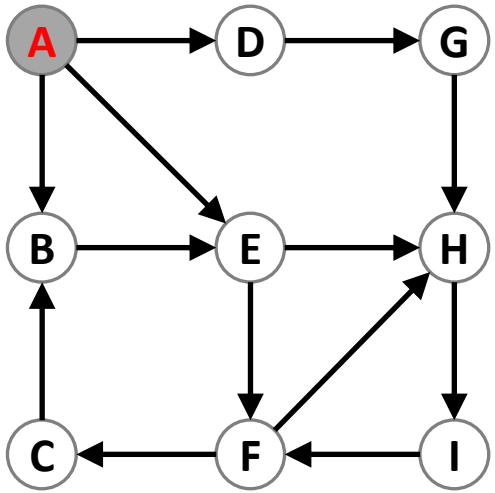
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A

Depth-First Traversal

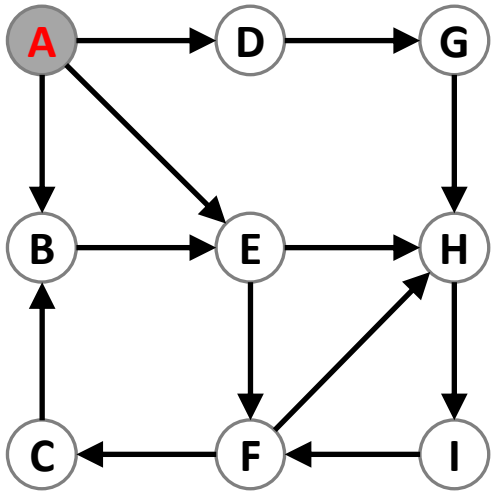
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	

Depth-First Traversal

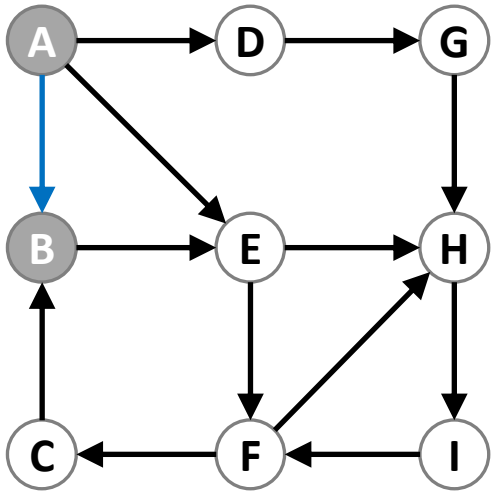
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B			

Depth-First Traversal

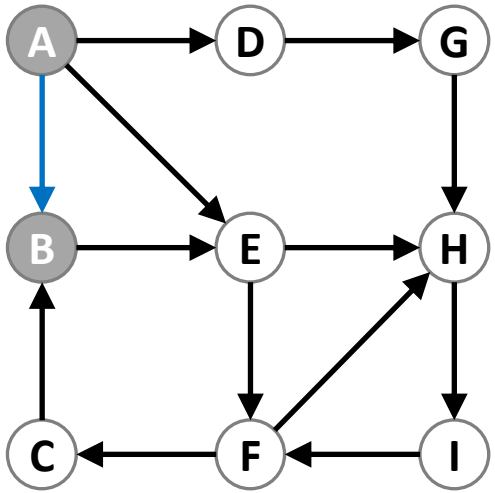
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB

Depth-First Traversal

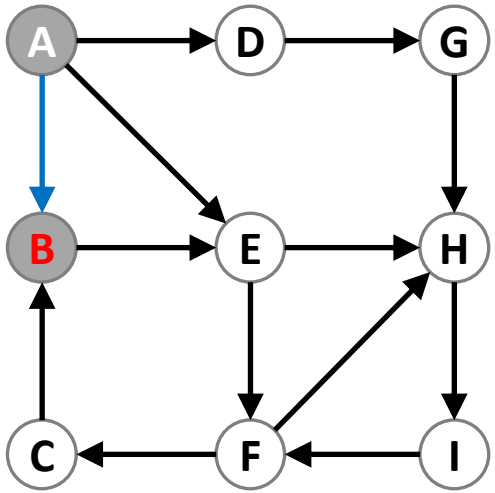
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	

Depth-First Traversal

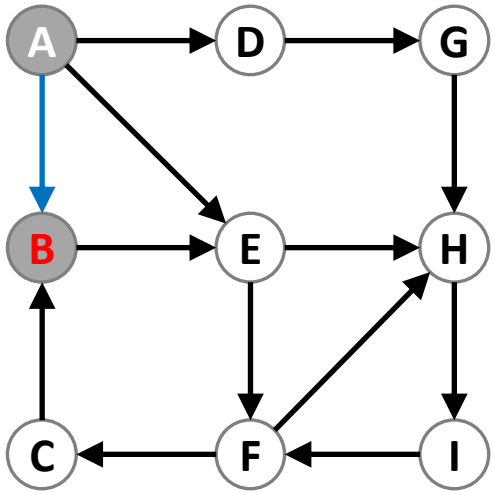
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	

Depth-First Traversal

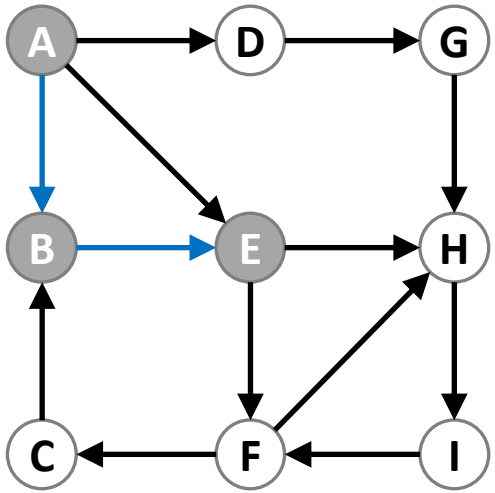
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E			

Depth-First Traversal

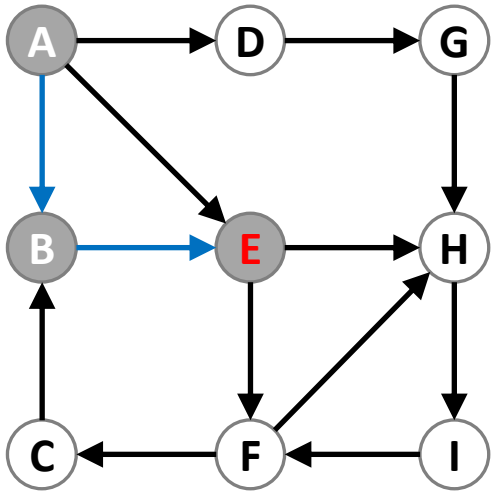
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE

Depth-First Traversal

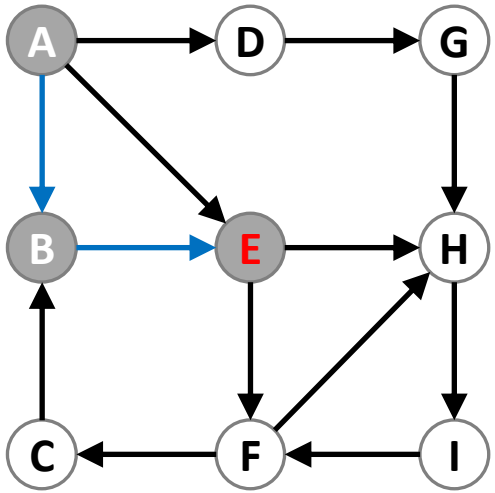
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	

Depth-First Traversal

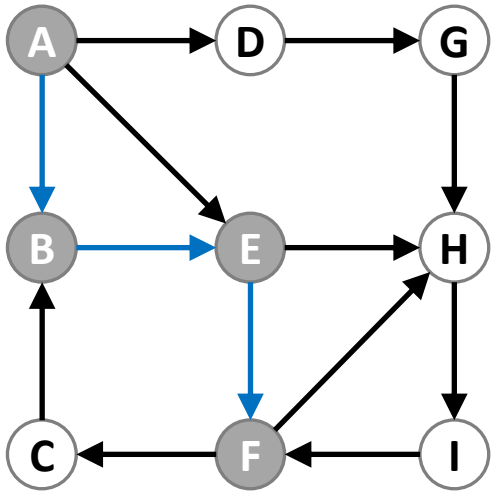
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F			

Depth-First Traversal

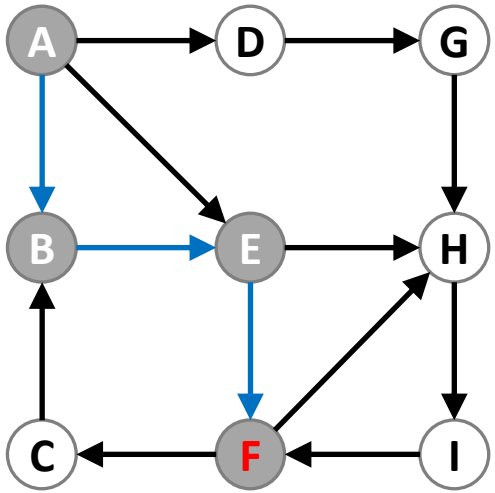
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF

Depth-First Traversal

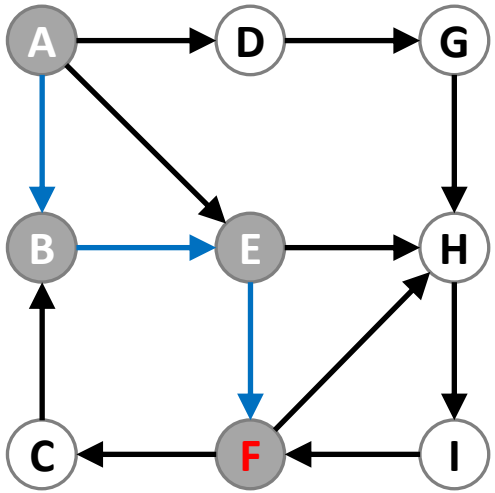
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	

Depth-First Traversal

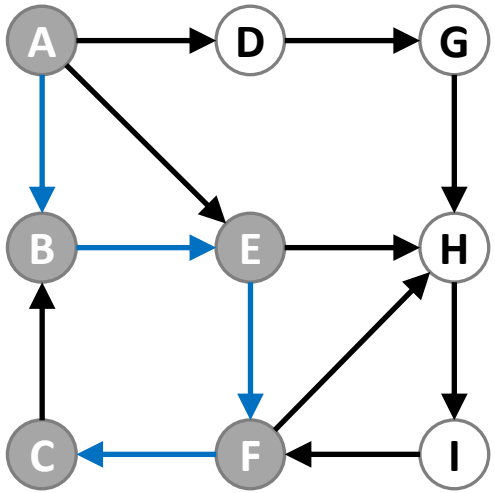
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C			

Depth-First Traversal

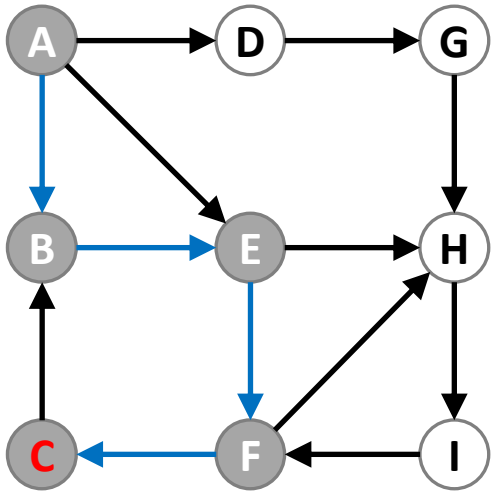
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC

Depth-First Traversal

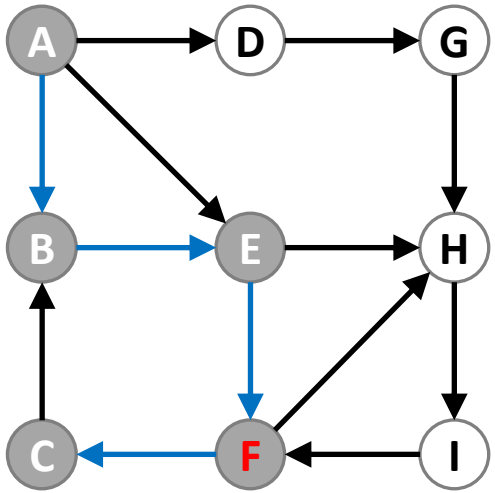
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	

Depth-First Traversal

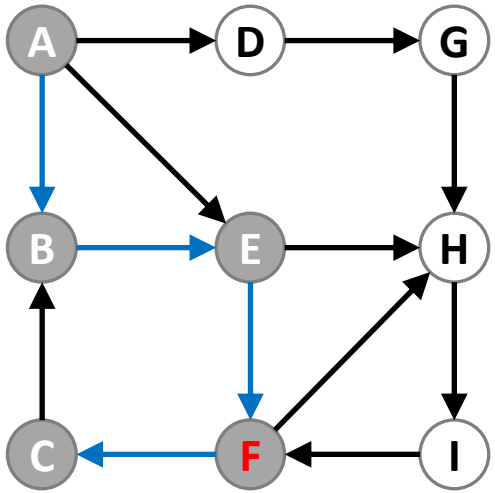
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	
F			FEBA	

Depth-First Traversal

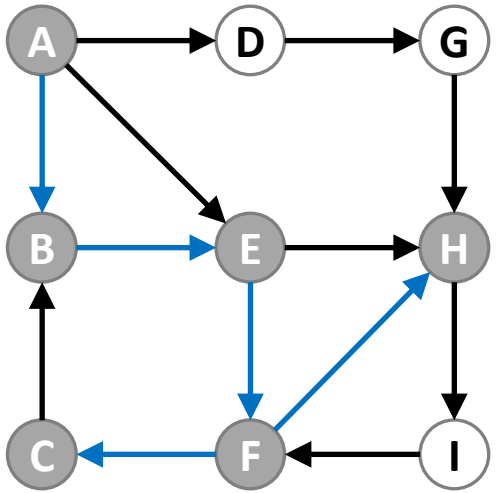
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	
F			FEBA	
	H			

Depth-First Traversal

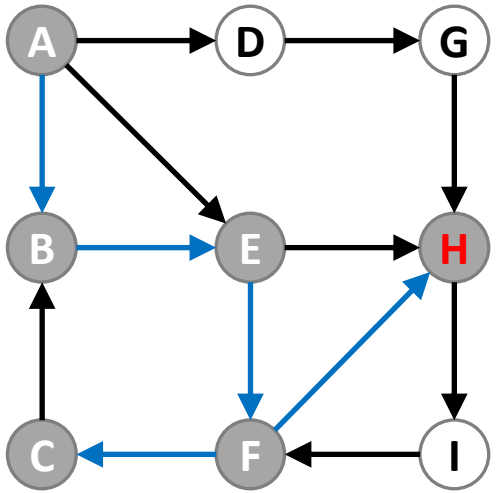
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	
F			FEBA	
	H	H	HFEBA	ABEFCH

Depth-First Traversal

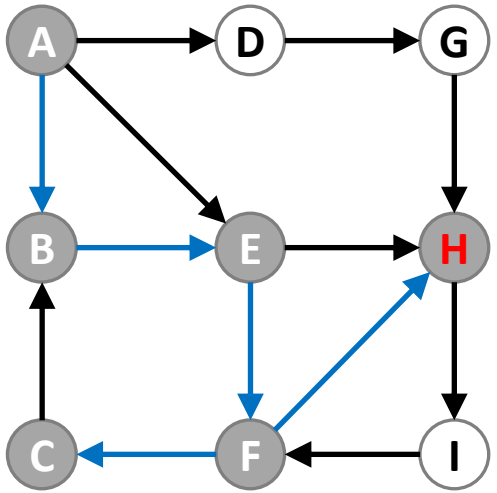
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	
F			FEBA	
	H	H	HFEBA	ABEFCH
H			HFEBA	

Depth-First Traversal

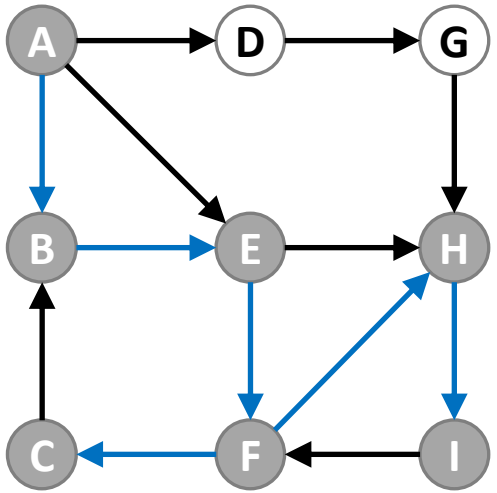
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	
F			FEBA	
	H	H	HFEBA	ABEFCH
H			HFEBA	
	I			

Depth-First Traversal

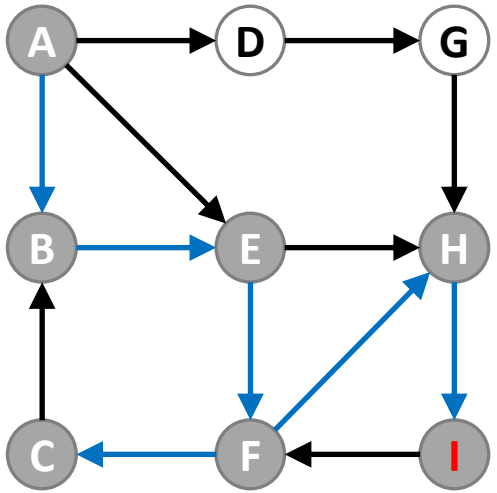
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	
F			FEBA	
	H	H	HFEBA	ABEFCH
H			HFEBA	
	I	I	IHFEBA	ABEFCHI

Depth-First Traversal

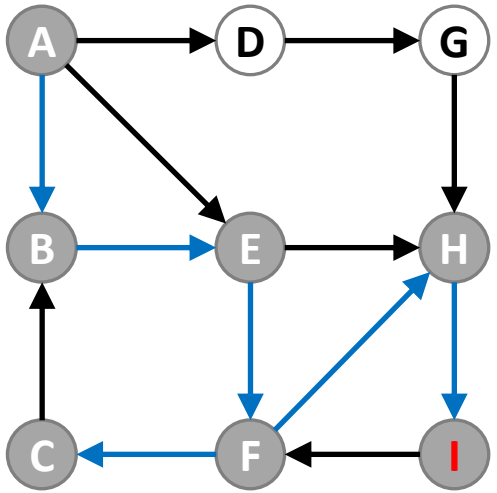
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	
F			FEBA	
	H	H	HFEBA	ABEFCH
H			HFEBA	
	I	I	IHFEBA	ABEFCHI
I				

Depth-First Traversal

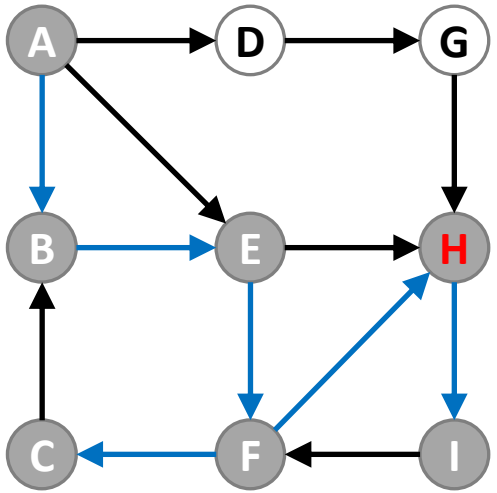
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	
F			FEBA	
	H	H	HFEBA	ABEFCH
H			HFEBA	
	I	I	IHFEBA	ABEFCHI
I			HFEBA	

Depth-First Traversal

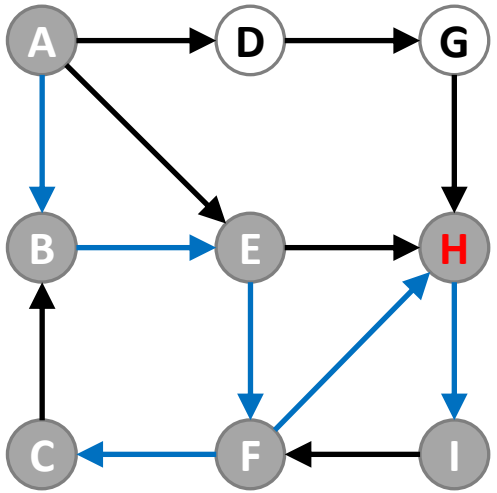
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	
F			FEBA	
	H	H	HFEBA	ABEFCH
H			HFEBA	
	I	I	IHFEBA	ABEFCHI
I			HFEBA	
H				

Depth-First Traversal

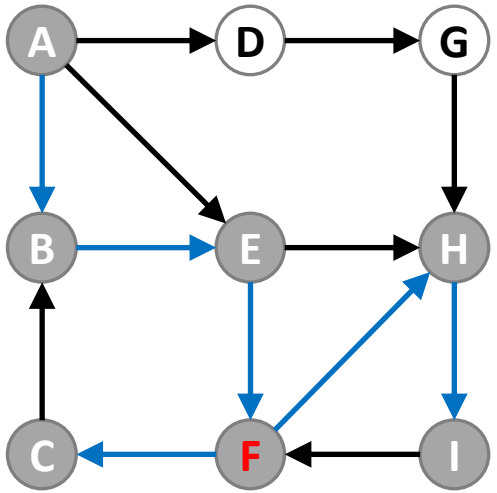
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	
F			FEBA	
	H	H	HFEBA	ABEFCH
H			HFEBA	
	I	I	IHFEBA	ABEFCHI
I			HFEBA	
H			FEBA	

Depth-First Traversal

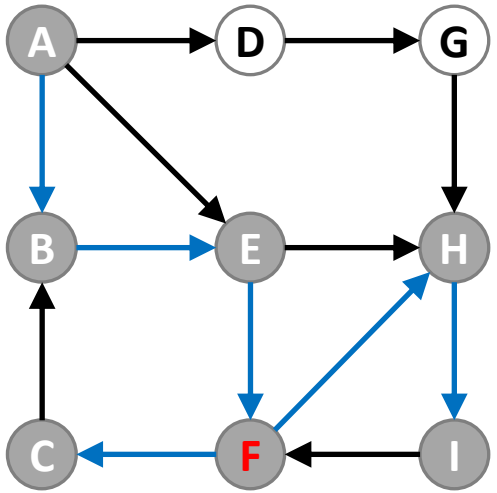
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	
F			FEBA	
	H	H	HFEBA	ABEFCH
H			HFEBA	
	I	I	IHFEBA	ABEFCHI
I			HFEBA	
H			FEBA	
F				

Depth-First Traversal

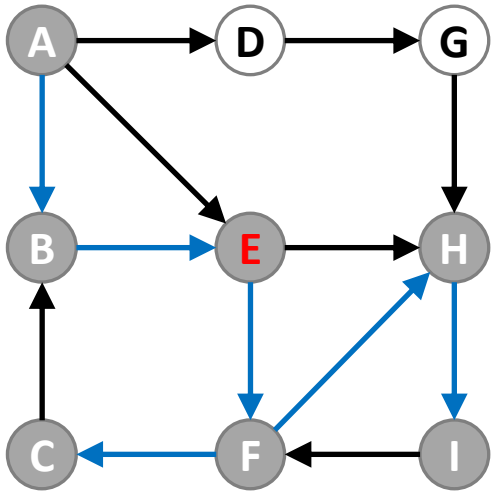
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	
F			FEBA	
	H	H	HFEBA	ABEFCH
H			HFEBA	
	I	I	IHFEBA	ABEFCHI
I			HFEBA	
H			FEBA	
F			EBA	

Depth-First Traversal

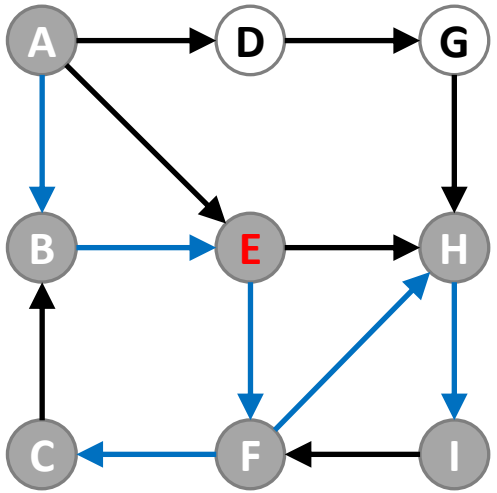
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	
F			FEBA	
	H	H	HFEBA	ABEFCH
H			HFEBA	
	I	I	IHFEBA	ABEFCHI
I			HFEBA	
H			FEBA	
F			EBA	
E				

Depth-First Traversal

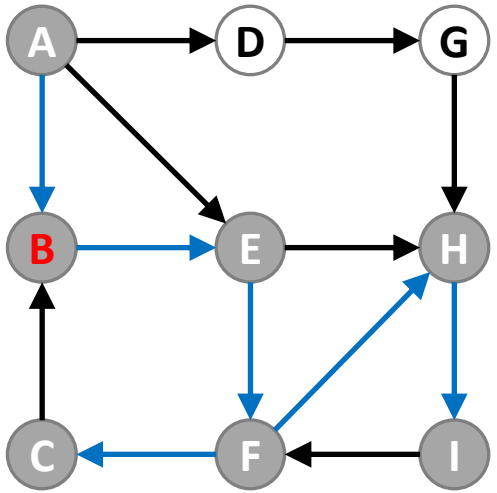
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	
F			FEBA	
	H	H	HFEBA	ABEFCH
H			HFEBA	
	I	I	IHFEBA	ABEFCHI
I			HFEBA	
H			FEBA	
F			EBA	
E			BA	

Depth-First Traversal

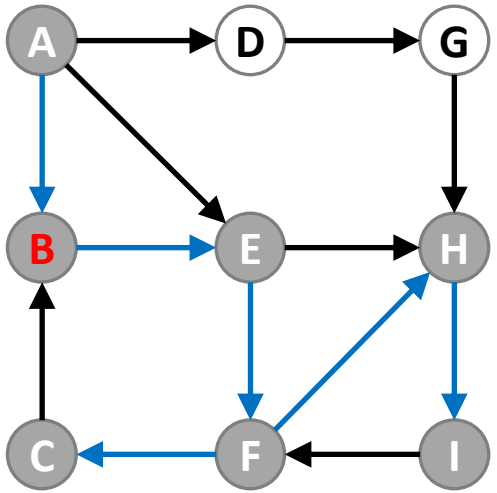
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	
F			FEBA	
	H	H	HFEBA	ABEFCH
H			HFEBA	
	I	I	IHFEBA	ABEFCHI
I			HFEBA	
H			FEBA	
F			EBA	
E			BA	
B				

Depth-First Traversal

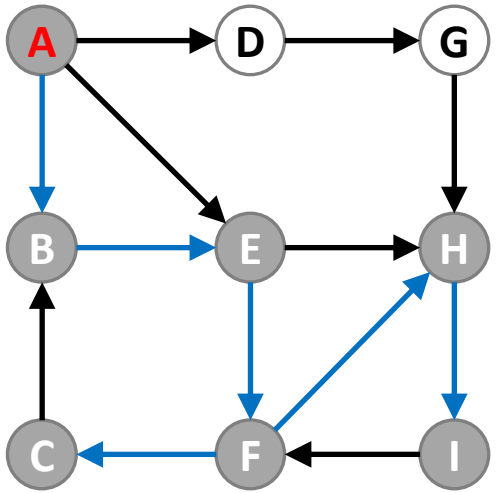
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	
F			FEBA	
	H	H	HFEBA	ABEFCH
H			HFEBA	
	I	I	IHFEBA	ABEFCHI
I			HFEBA	
H			FEBA	
F			EBA	
E			BA	
B			A	

Depth-First Traversal

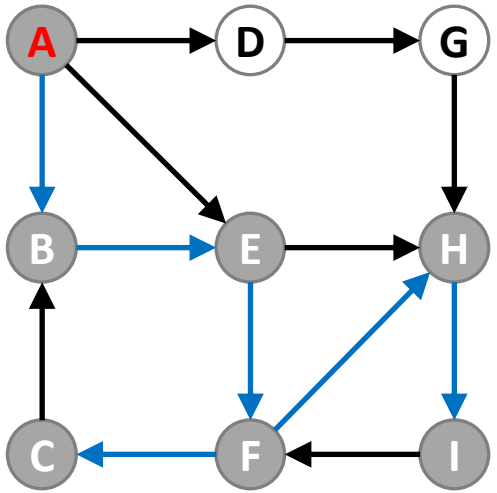
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	
F			FEBA	
	H	H	HFEBA	ABEFCH
H			HFEBA	
	I	I	IHFEBA	ABEFCHI
I			HFEBA	
H			FEBA	
F			EBA	
E			BA	
B			A	
A				

Depth-First Traversal

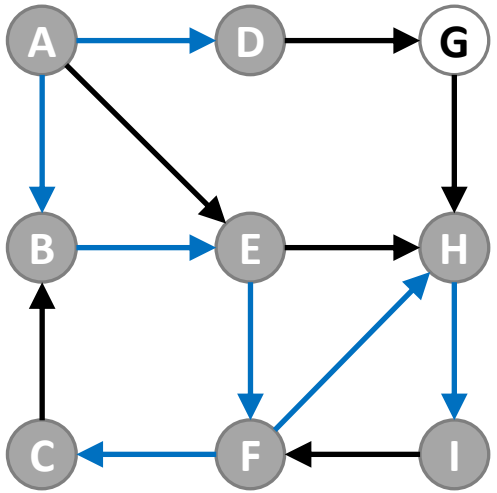
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	
F			FEBA	
	H	H	HFEBA	ABEFCH
H			HFEBA	
	I	I	IHFEBA	ABEFCHI
I			HFEBA	
H			FEBA	
F			EBA	
E			BA	
B			A	
A			A	
	D			

Depth-First Traversal

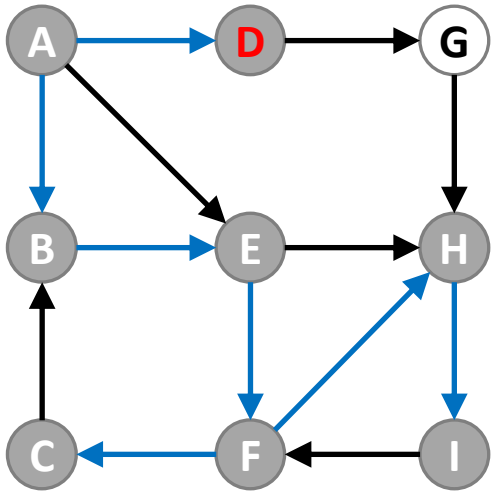
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	
F			FEBA	
	H	H	HFEBA	ABEFCH
H			HFEBA	
	I	I	IHFEBA	ABEFCHI
I			HFEBA	
H			FEBA	
F			EBA	
E			BA	
B			A	
A			A	
	D	D	DA	ABEFCHID

Depth-First Traversal

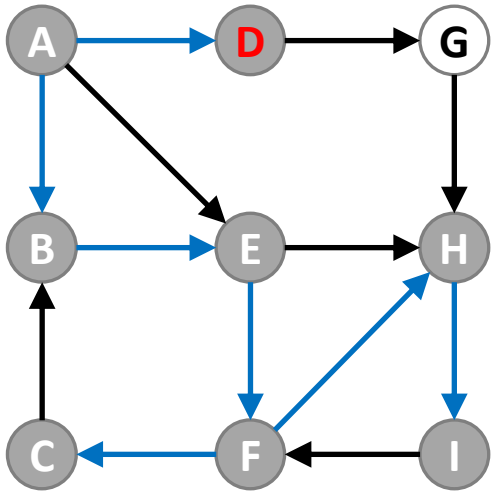
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	
F			FEBA	
	H	H	HFEBA	ABEFCH
H			HFEBA	
	I	I	IHFEBA	ABEFCHI
I			HFEBA	
H			FEBA	
F			EBA	
E			BA	
B			A	
A			A	
	D	D	DA	ABEFCHID
D				

Depth-First Traversal

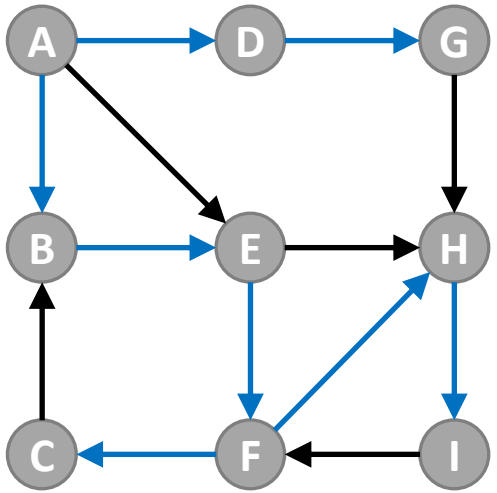
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	
F			FEBA	
	H	H	HFEBA	ABEFCH
H			HFEBA	
	I	I	IHFEBA	ABEFCHI
I			HFEBA	
H			FEBA	
F			EBA	
E			BA	
B			A	
A			A	
	D	D	DA	ABEFCHID
D			DA	
	G			

Depth-First Traversal

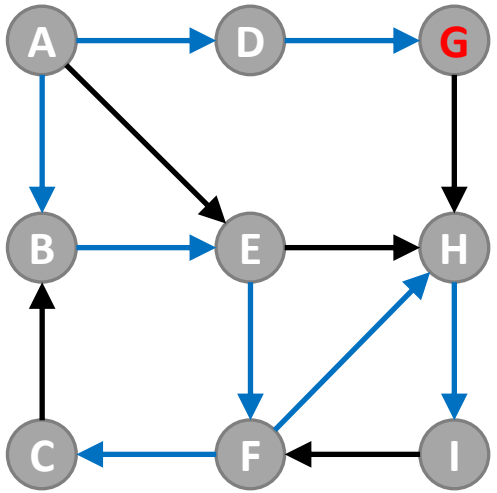
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	
F			FEBA	
	H	H	HFEBA	ABEFCH
H			HFEBA	
	I	I	IHFEBA	ABEFCHI
I			HFEBA	
H			FEBA	
F			EBA	
E			BA	
B			A	
A			A	
	D	D	DA	ABEFCHID
D			DA	
	G	G	GDA	ABEFCHIDG

Depth-First Traversal

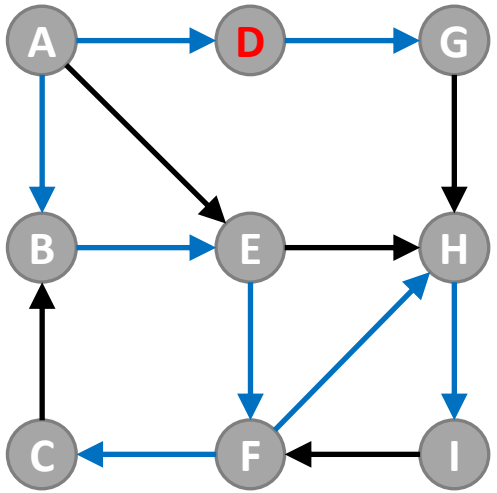
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	
F			FEBA	
	H	H	HFEBA	ABEFCH
H			HFEBA	
	I	I	IHFEBA	ABEFCHI
I			HFEBA	
H			FEBA	
F			EBA	
E			BA	
B			A	
A			A	
	D	D	DA	ABEFCHID
D			DA	
	G	G	GDA	ABEFCHIDG
G			DA	

Depth-First Traversal

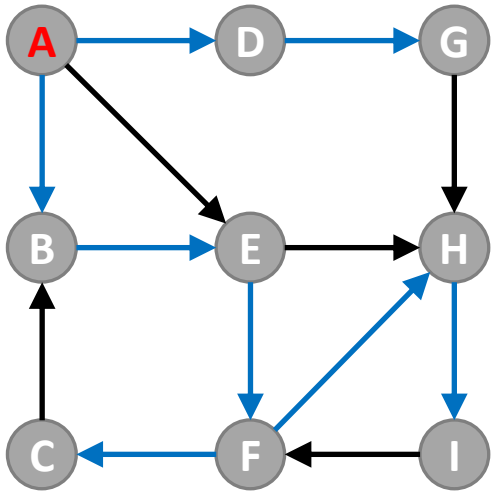
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.



topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	
F			FEBA	
	H	H	HFEBA	ABEFCH
H			HFEBA	
	I	I	IHFEBA	ABEFCHI
I			HFEBA	
H			FEBA	
F			EBA	
E			BA	
B			A	
A			A	
	D	D	DA	ABEFCHID
D			DA	
	G	G	GDA	ABEFCHIDG
G			DA	
D			A	

Depth-First Traversal

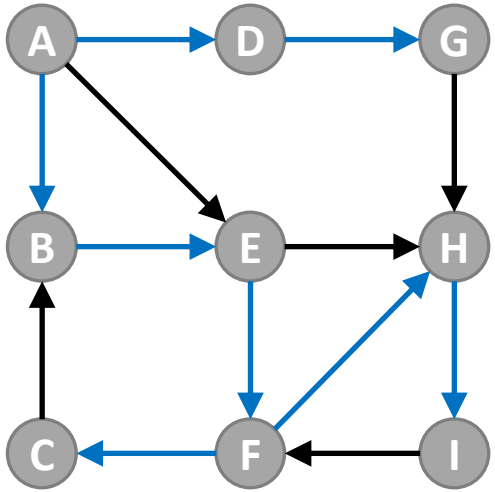
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.

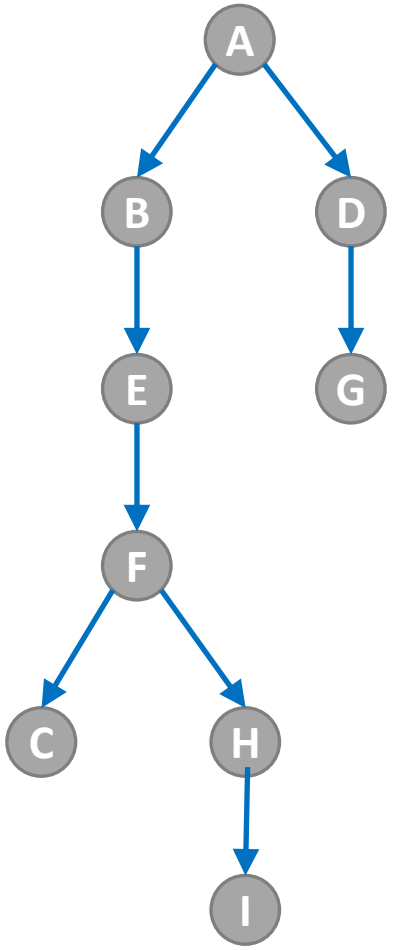


topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	
F			FEBA	
	H	H	HFEBA	ABEFCH
H			HFEBA	
	I	I	IHFEBA	ABEFCHI
I			HFEBA	
H			FEBA	
F			EBA	
E			BA	
B			A	
A			A	
	D	D	DA	ABEFCHID
D			DA	
	G	G	GDA	ABEFCHIDG
G			DA	
D			A	
A			*empty*	ABEFCHIDG

Depth-First Traversal

- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backs up by one vertex, it considers another neighbor.

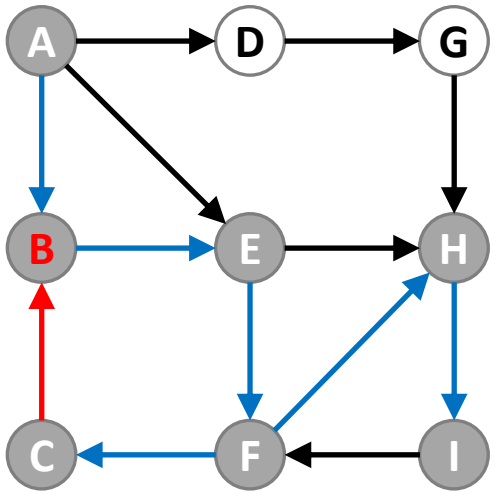




Paths form a depth-first tree
(Order in which the nodes are expanded)

Depth-First Traversal

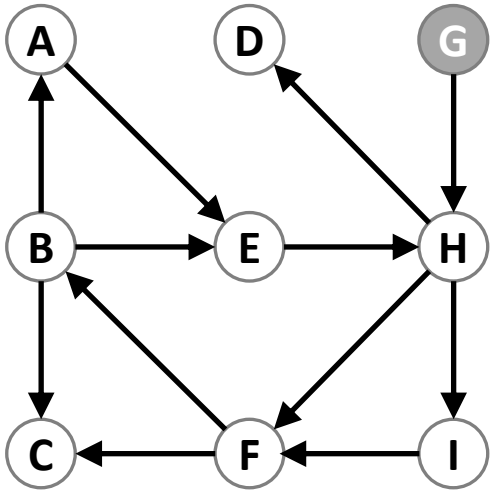
- Depth First Traversal can be used to detect cycle in a Graph



http://www.geeksforgeeks.org/detect-cycle-in-a-graph/

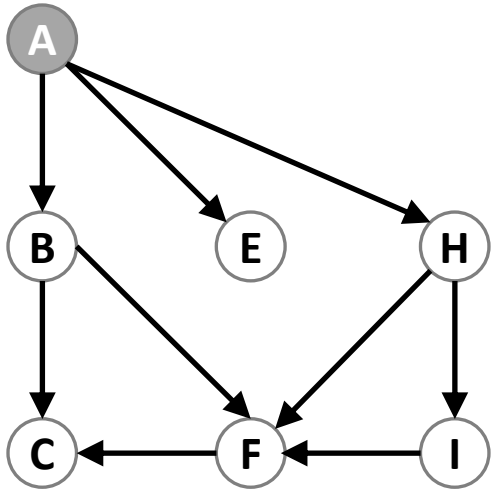
In-Class Exercise

- Perform depth-first traversal beginning with Vertex *G*



In-Class Exercise

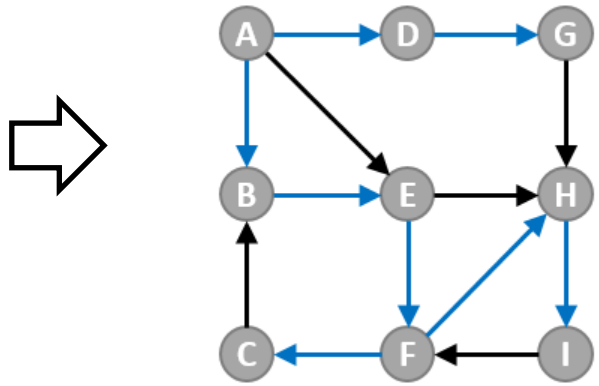
- Perform depth-first traversal beginning with Vertex *A*



Implementation of DFS

- Depth-first search uses a stack (can also implement it recursively) to expand the deepest unvisited nodes.

```
Algorithm getDepthFirstTraversal(originVertex)
traversalOrder = a new queue for the resulting traversal order
vertexStack = a new stack to hold vertices as they are visited

Mark originVertex as visited
traversalOrder.enqueue(originVertex)
vertexStack.push(originVertex)

while (!vertexStack.isEmpty())
{
   topVertex = vertexStack.peek()
   if (topVertex has an unvisited neighbor)
   {
      nextNeighbor = next unvisited neighbor of topVertex
      Mark nextNeighbor as visited
      traversalOrder.enqueue(nextNeighbor)
      vertexStack.push(nextNeighbor)
   }
   else // all neighbors are visited
      vertexStack.pop()
}
return traversalOrder
```

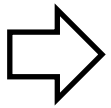


topVertex	nextNeighbor	Visited vertex	vertexStack (top to bottom)	traversalOrder (front to back)
		A	A	A
A			A	
	B	B	BA	AB
B			BA	
	E	E	EBA	ABE
E			EBA	
	F	F	FEBA	ABEF
F			FEBA	
	C	C	CFEBA	ABEFC
C			FEBA	
F			FEBA	
	H	H	HFEBA	ABEFCH
H			HFEBA	
	I	I	IHFEBA	ABEFCHI
I			HFEBA	
H			FEBA	
F			EBA	
E			BA	
B			A	
A			A	
	D	D	DA	ABEFCHID
D			DA	
	G	G	GDA	ABEFCHIDG
G			DA	
D			A	
A			*empty*	ABEFCHIDG

Implementation of DFS

- Depth-first search uses a stack (can also implement it recursively) to expand the deepest unvisited nodes.

```
Algorithm getDepthFirstTraversal(originVertex)
traversalOrder = a new queue for the resulting traversal order
vertexStack = a new stack to hold vertices as they are visited

Mark originVertex as visited
traversalOrder.enqueue(originVertex)
vertexStack.push(originVertex)

while (!vertexStack.isEmpty())
{
   topVertex = vertexStack.peek()
   if (topVertex has an unvisited neighbor)
   {
      nextNeighbor = next unvisited neighbor of topVertex
      Mark nextNeighbor as visited
      traversalOrder.enqueue(nextNeighbor)
      vertexStack.push(nextNeighbor)
   }
   else // all neighbors are visited
      vertexStack.pop()
}
return traversalOrder
```



```
public QueueInterface<T> getDepthFirstTraversal(T origin)
{
    // Assumes graph is not empty
    resetVertices();
    QueueInterface<T> traversalOrder = new LinkedQueue<T>();
    StackInterface<VertexInterface<T>> vertexStack = new LinkedStack<>();

    VertexInterface<T> originVertex = vertices.getValue(origin);
    originVertex.visit();
    traversalOrder.enqueue(origin); // Enqueue vertex label
    vertexStack.push(originVertex); // Enqueue vertex

    while (!vertexStack.isEmpty())
    {
        VertexInterface<T> topVertex = vertexStack.peek();
        VertexInterface<T> nextNeighbor = topVertex.getUnvisitedNeighbor();

        if (nextNeighbor != null)
        {
            nextNeighbor.visit();
            traversalOrder.enqueue(nextNeighbor.getLabel());
            vertexStack.push(nextNeighbor);
        }
        else // All neighbors are visited
            vertexStack.pop();
    } // end while

    return traversalOrder;
} // end getDepthFirstTraversal
```

Implementation of DFS

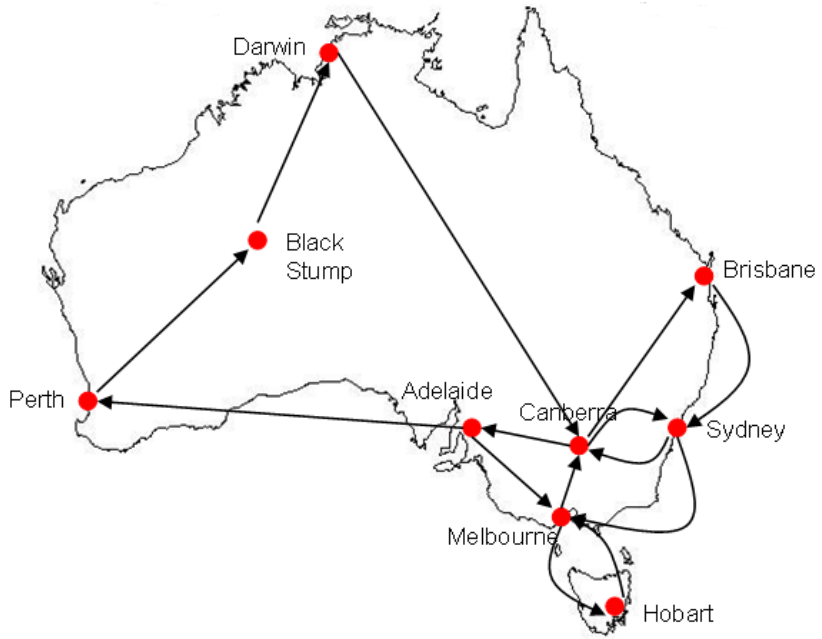
- A recursive implementation of DFS

```
1  procedure DFS(G,v):
2      label v as discovered
3      for all edges from v to w in G.adjacentEdges(v) do
4          if vertex w is not labeled as discovered then
5              recursively call DFS(G,w)
```


https://en.wikipedia.org/wiki/Depth-first_search

In-Class Exercise

- Perform DFS and BFS of the Australia graph starting at Sydney, and list the order in which the cities are visited.



Summary

- Graph Terminology
- Graph Representations
  - Adjacency matrix
  - Adjacency list
- Graph Traversals
  - Breadth-first search
  - Depth-first search

What I Want You to Do

- Review Class Slides
- Review Chapters 29 and 30