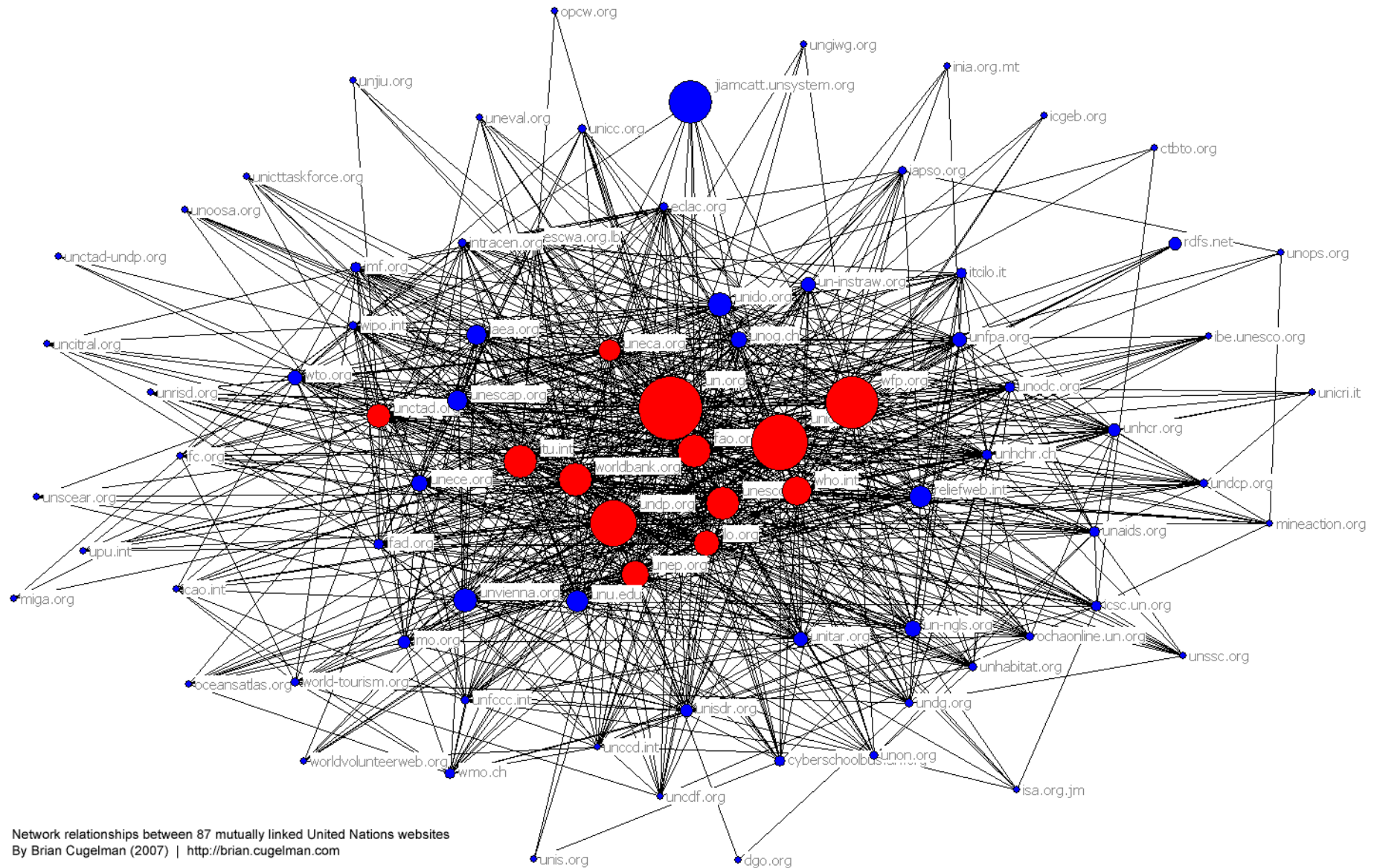


Introduction to Graphs

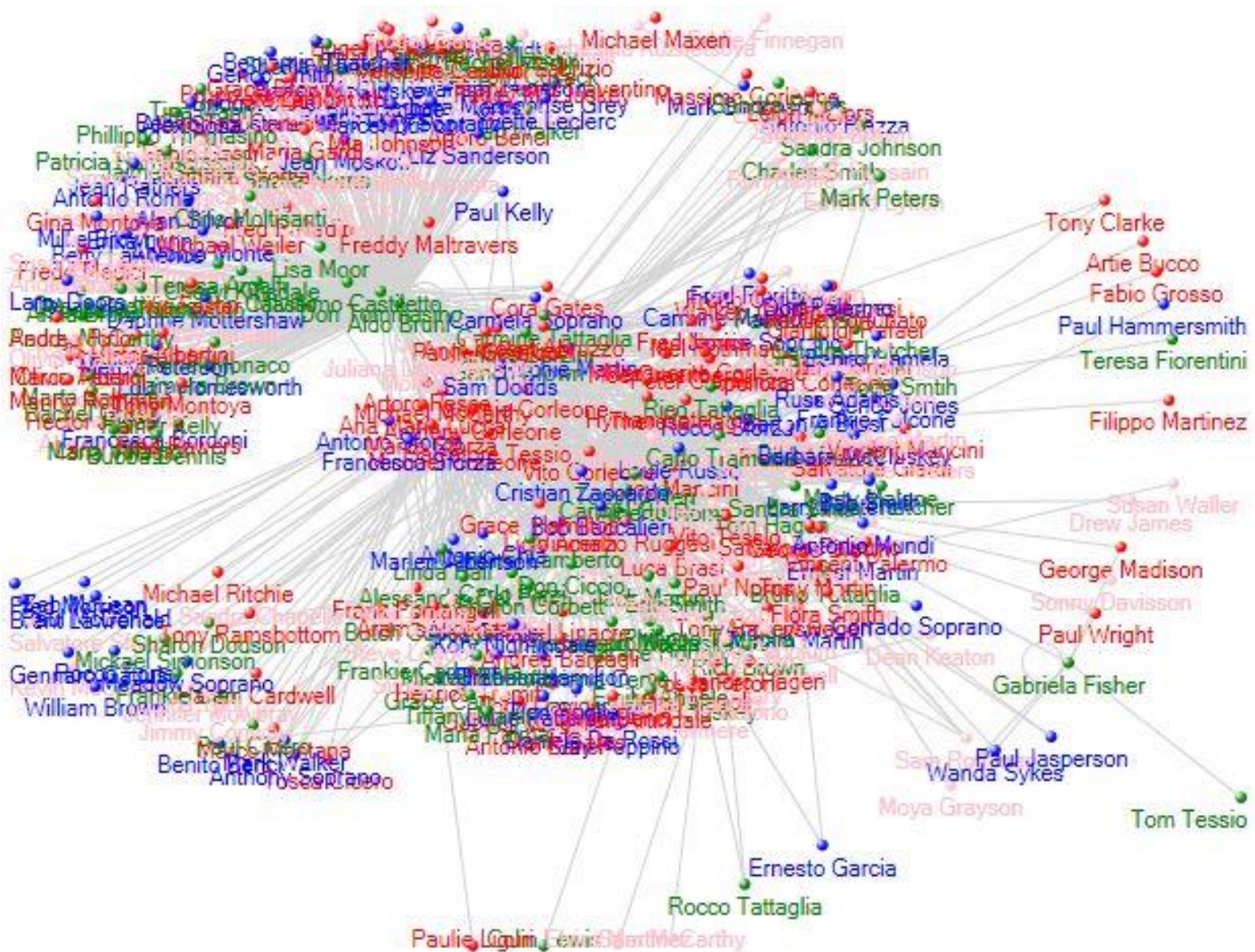
Outline

- Introduction & Terminology
- Representing Graphs
- Graph Traversals
- Spanning Trees
- Shortest Path Algorithms

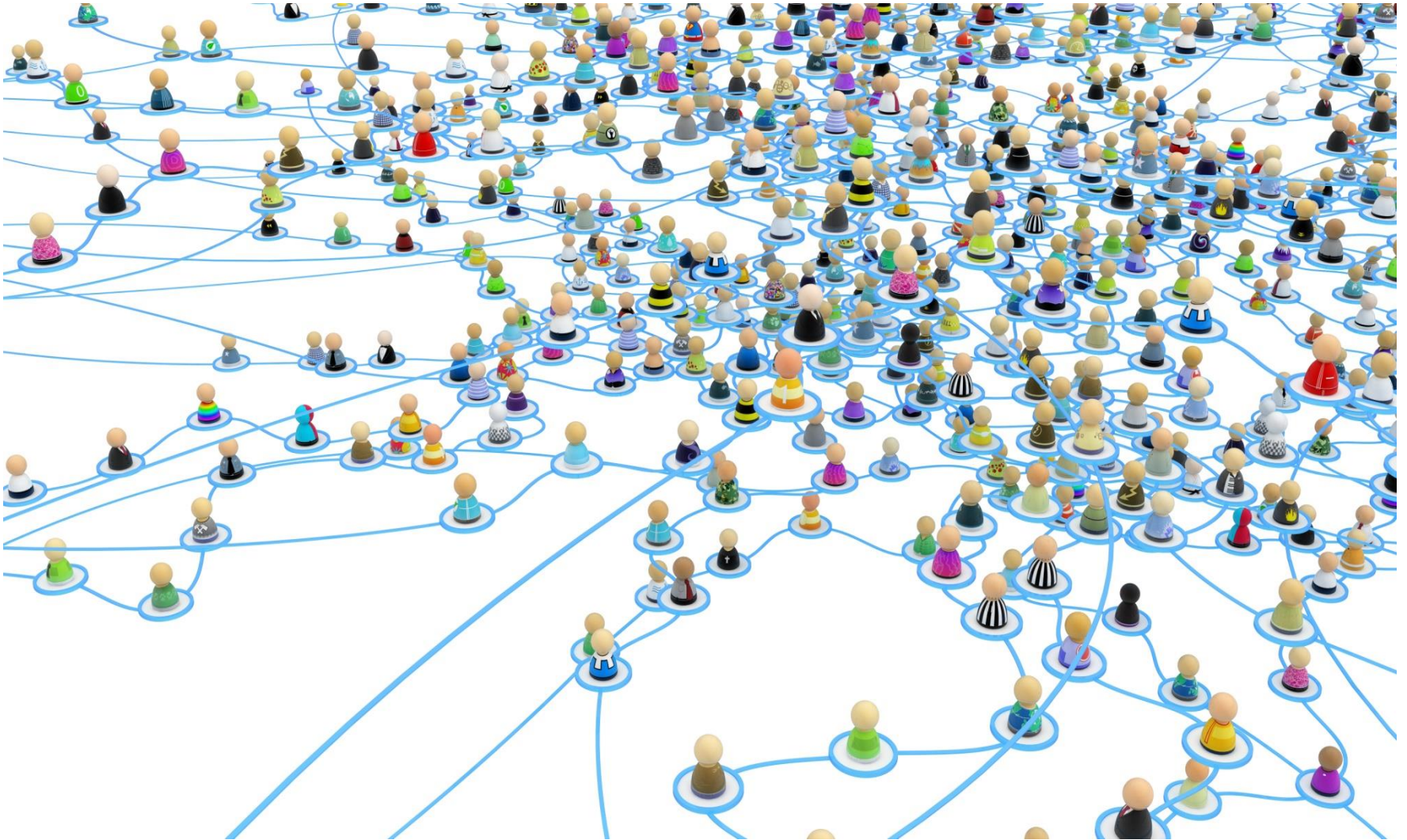
Internet



Email Networks



Social Networks

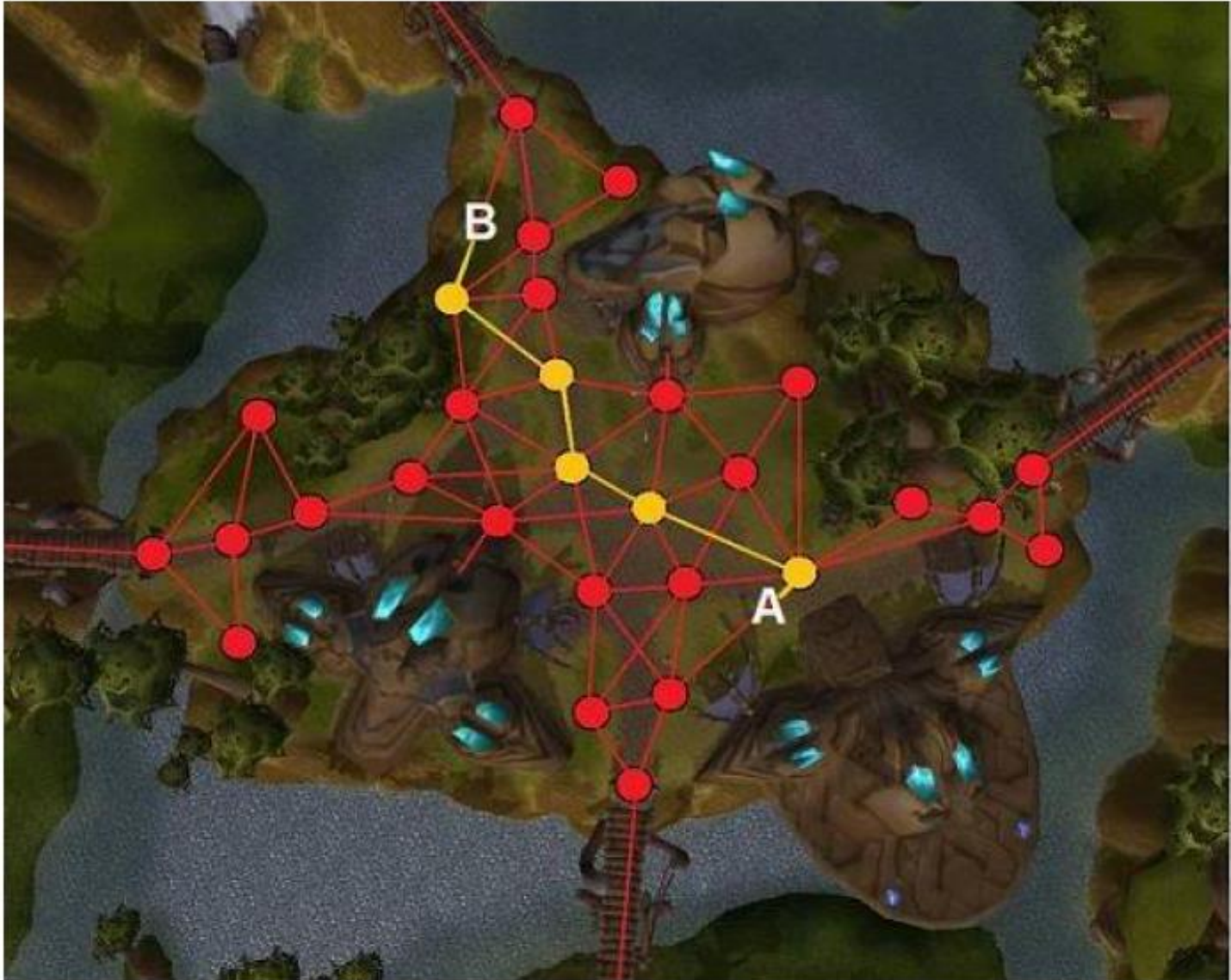




Video Games



Video Games



Introduction & Terminology

Introduction

- One of the most useful data structures (A very large topic).
- Related to trees in that a tree is a special kind of graph (Trees are much simpler).
- Graphs are more general and have a wider range of use. (Generality trades simplicity).
- Represent problems involving interconnected (dependent) objects.
- Graph algorithms are complex. Need to account for cycles; trees have only one path between nodes.

Introduction

- One of the most useful data structures (A very large topic).
- **Tree is a special kind of graph (Trees are much simpler).**
- Graphs are more general and have a wider range of use. (Generality trades simplicity).
- Represent problems involving interconnected (dependent) objects.
- Graph algorithms are complex. Need to account for cycles; trees have only one path between nodes.

Introduction

- One of the most useful data structures (A very large topic).
- Related to trees in that a tree is a special kind of graph (Trees are much simpler).
- **Graphs are more general and have a wider range of use. (Generality trades simplicity).**
- Represent problems involving interconnected (dependent) objects.
- Graph algorithms are complex. Need to account for cycles; trees have only one path between nodes.

Introduction

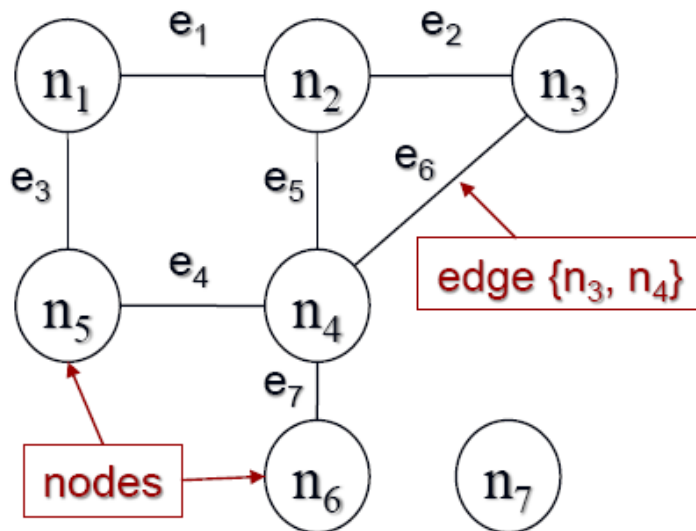
- One of the most useful data structures (A very large topic).
- Related to trees in that a tree is a special kind of graph (Trees are much simpler).
- Graphs are more general and have a wider range of use. (Generality trades simplicity).
- **Represent problems involving interconnected (dependent) objects.**
- Graph algorithms are complex. Need to account for cycles; trees have only one path between nodes.

Introduction

- One of the most useful data structures (A very large topic).
- Related to trees in that a tree is a special kind of graph (Trees are much simpler).
- Graphs are more general and have a wider range of use. (Generality trades simplicity).
- Represent problems involving interconnected (dependent) objects.
- Graph algorithms are complex. Need to account for cycles; trees have only one path between nodes.

Terminology

- A graph is essentially a **collection of points** connected by **line segments**.
- The points are referred to as **nodes** or **vertices**; the segments are called **edges**.



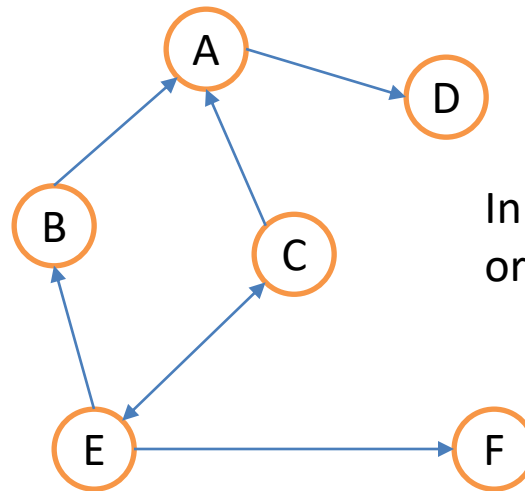
$$V = \{n_1, n_2, n_3, n_4, n_5, n_6, n_7\}$$

$$E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7\}$$

$$= \{(n_1, n_2), (n_2, n_3), (n_1, n_5), (n_4, n_5), (n_2, n_4), (n_3, n_4), (n_4, n_6)\}$$

Terminology

- If the edges have a direction (arrowheads in a diagram), the graph is a **directed graph**, or **digraph**.

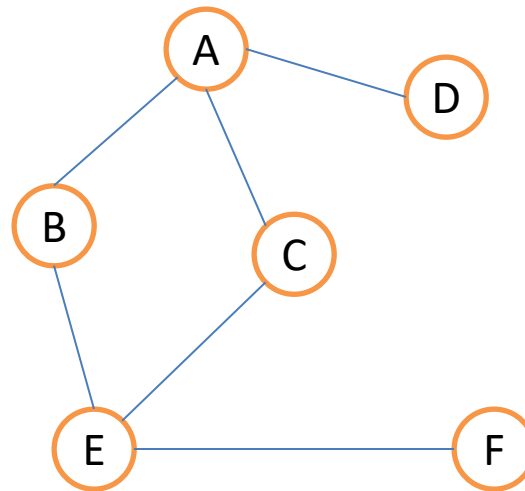


In a **directed graph**, each edge is an ordered pair: $e=(C,A)$

C is the origin (source) and A is the terminus (destination).

Terminology

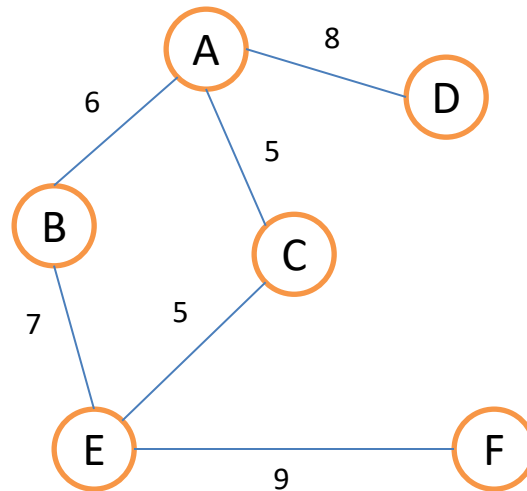
- If the graph edges has no values is called a **undirected graph**.



In an **undirected graph**, each edge is an unordered pair: $e=(v_1,v_2)$

Terminology

- If the graph has values (**weights** or **costs**) assigned to edges is called a **weighted graph**.



Notation

- Two vertices, x and y are said to be **adjacent** if there is an edge connecting them.
- We use the notation sGd to mean that s is adjacent to d . With a digraph, sGd implies direction. (xGy is not the same as yGx).
- The set of nodes adjacent to s is called the adjacency set of s or neighbors of s .
 - This set is fundamental to many graph algorithms.

Notation

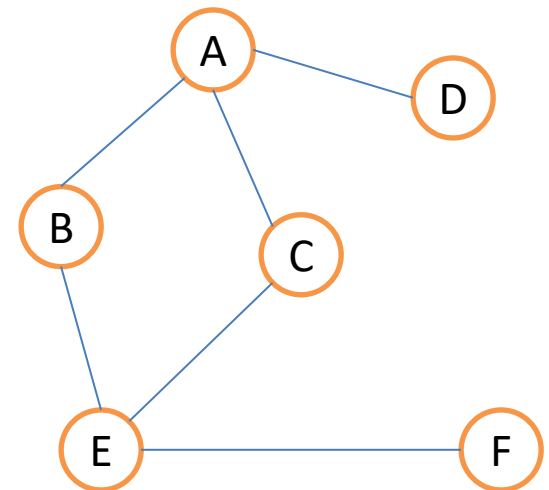
- Two vertices, x and y are said to be adjacent if there is an edge connecting them.
- We use the notation sGd to mean that s is adjacent to d . With a digraph, sGd implies direction. (xGy is not the same as yGx).
- The set of nodes adjacent to s is called the adjacency set of s or neighbors of s .
 - This set is fundamental to many graph algorithms.

Notation

- Two vertices, x and y are said to be adjacent if there is an edge connecting them.
- We use the notation sGd to mean that s is adjacent to d . With a digraph, sGd implies direction. (xGy is not the same as yGx).
- The set of nodes adjacent to s is called the **adjacency set** of s or **neighbors** of s .
 - This set is fundamental to many graph algorithms.

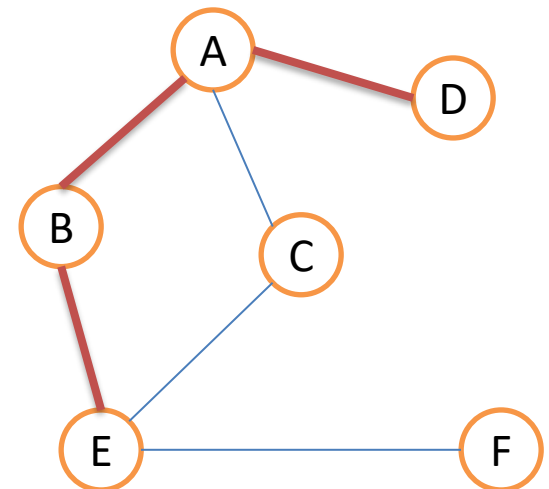
Paths and Connectivity

- A (contiguous) sequence of edges is a path



Paths and Connectivity

- A (contiguous) sequence of edges is a path
path D to E : {D, A, B, E}
- If there is a path from x to y, y is **reachable** from x
- The length of a path is the number of edges on the path: 3 for the example
- Two vertices are connected if there is a path from one to the other
- A **connected component** is a subset, S, of vertices that are all connected

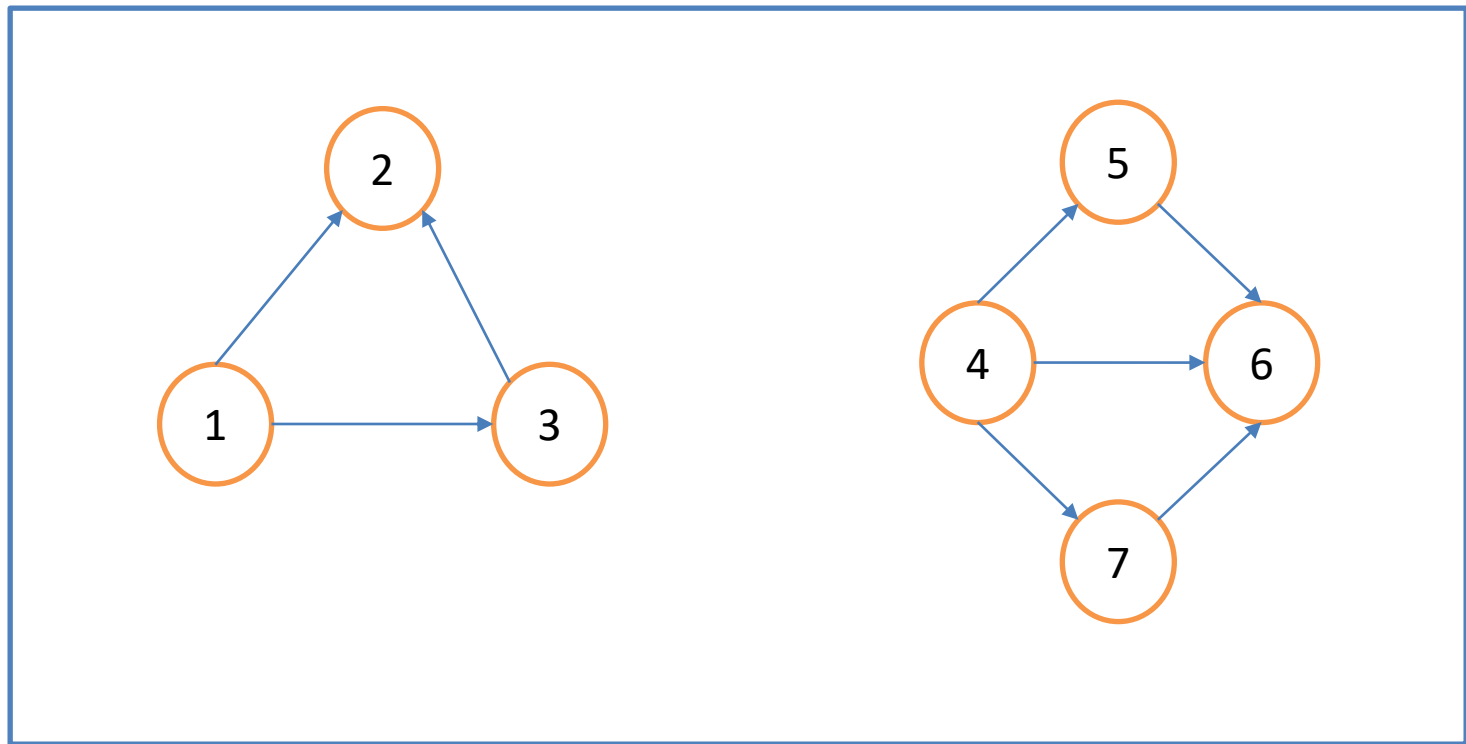


Connectedness

- Connectedness is an equivalence relation on the node set of a graph
 - **Reflexive**: every node is in a path of length 0 with itself
 - **Symmetric**: if $(n_i, n_j) \in \text{path}$, then $(n_j, n_i) \in \text{path}$
 - **Transitive**: if $(n_i, n_j) \in \text{path}$ and $(n_j, n_w) \in \text{path}$, then $(n_i, n_w) \in \text{path}$.

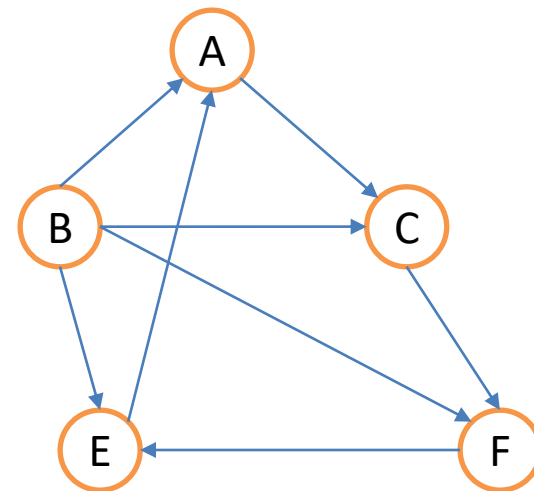
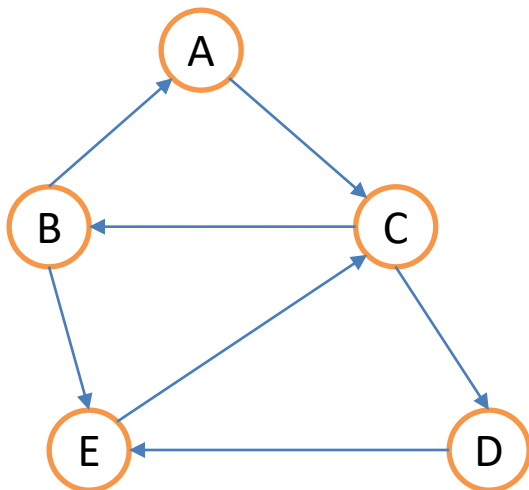
Connected Component: Example

- A single directed graph with two components:



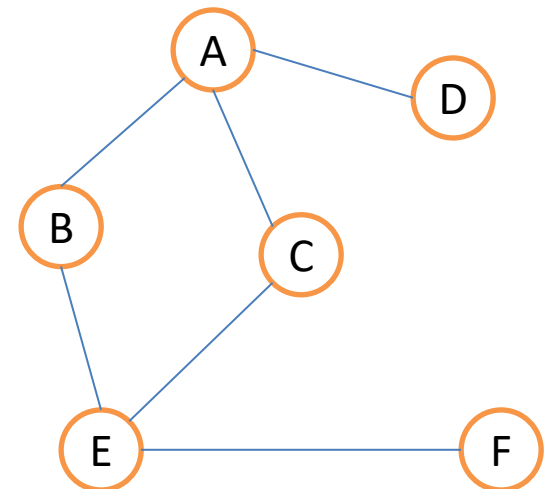
Connection Types

- Digraphs can be strongly connected or weakly connected.
 - **Strongly connected:** There is a path from every node to every other node
 - **Weakly connected:** There is **NOT** a path from each node to every other node (see Node C)



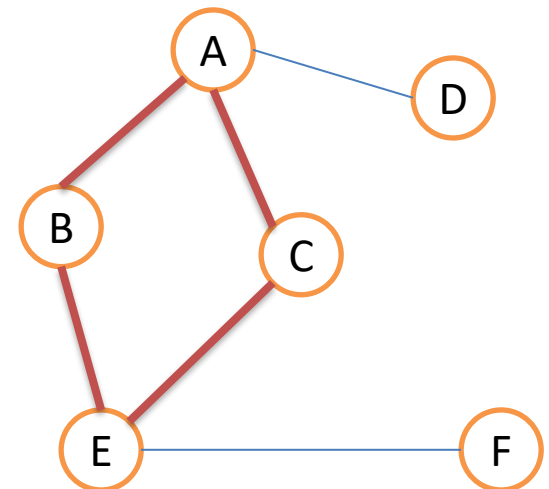
Cycles

- A cycle is a path whose source and destination node are the same
- A cycle is **simple** if all nodes on the path are distinct (with the exception of the first and last). A simple cycle must include at least 3 vertices



Cycles

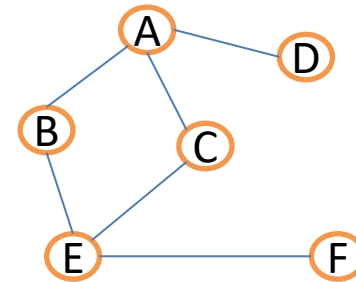
- A cycle is a path whose source and destination node are the same
- A cycle is **simple** if all nodes on the path are distinct (with the exception of the first and last). A simple cycle must include at least 3 vertices
- If a graph has no cycles, it is **acyclic**. A directed acyclic graph is called a **DAG**



Degree

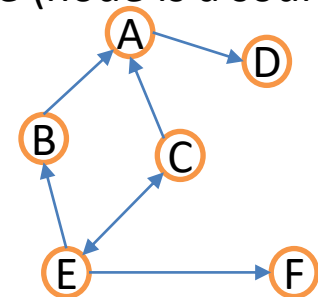
- For an undirected graph, the degree is the number of edges connecting a node

- A: 3
- B: 2



- For directed graph:

- In-degree: Is the number of incoming edges into a node (node is a destination).
 - A:2
 - B:1
- Out-degree: Is the number of outgoing edges from a node (node is a source).
 - A:1
 - B:1



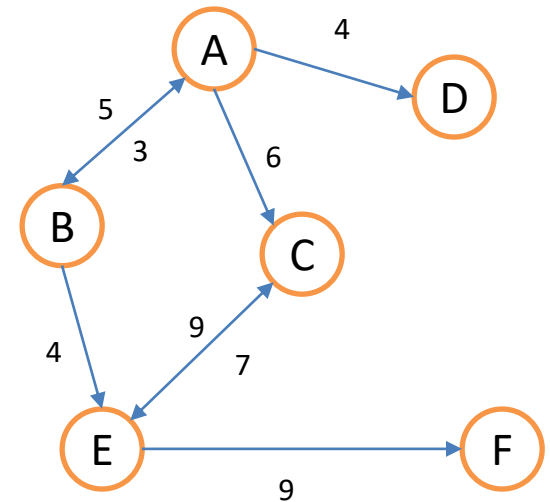
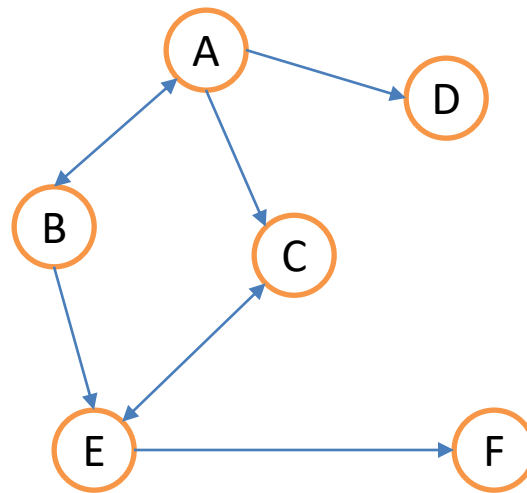
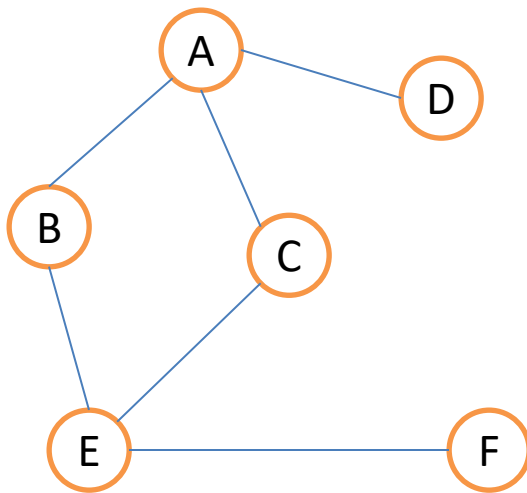
Tree vs Graph

Characteristics	Tree	Graph
Cycle	X	✓
Root	✓	X
Reachability of Nodes	✓	X
Direction	✓	Optional

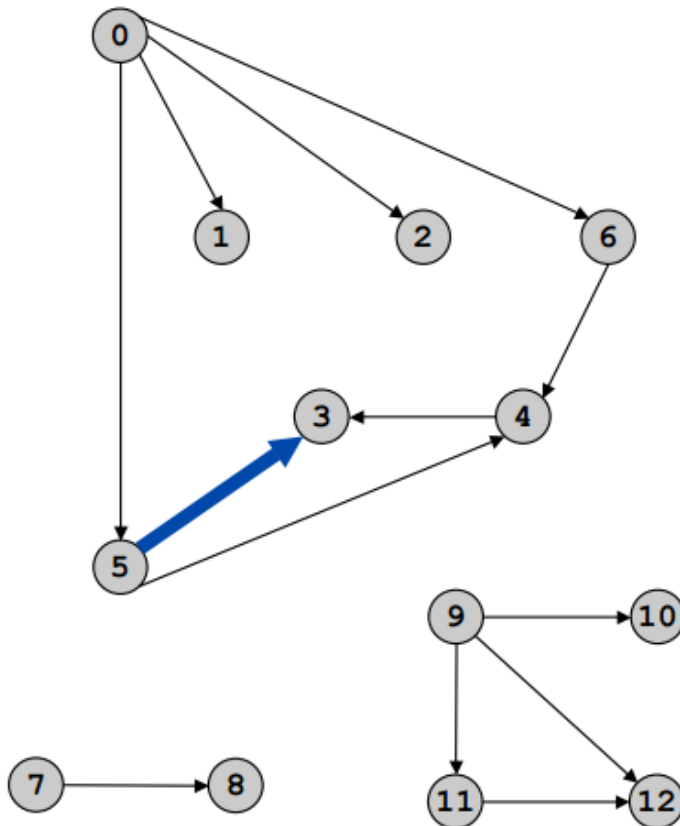
Representing Graphs

Adjacency Matrix

- A graph G with N nodes represented by an $N \times N$ boolean array (matrix).
- For each x and y , $G(x,y) = \text{TRUE}$ if xGy , otherwise false.



Adjacency Matrix - Example

[illegible]

Adjacency Matrix

- Space required is $O(N^2)$
 - A sparse graph has few edges
 - Sparse graphs will have many matrix entries of 0.
 - a dense graph has many edges.
 - Dense graphs will have many matrix entries of 1.
- Determining if two nodes are adjacent is $O(1)$
- The size of the matrix is **independent** of the number of edges.
- An adjacency matrix may be a more desirable representation for dense graphs.

Adjacency Matrix

- Space required is $O(N^2)$
 - A **sparse** graph has few edges
 - Sparse graphs will have many matrix entries of 0.
 - a dense graph has many edges.
 - Dense graphs will have many matrix entries of 1.
- Determining if two nodes are adjacent is $O(1)$
- The size of the matrix is **independent** of the number of edges.
- An adjacency matrix may be a more desirable representation for dense graphs.

Adjacency Matrix

- Space required is $O(N^2)$
 - A **sparse** graph has few edges
 - Sparse graphs will have many matrix entries of 0.
 - a **dense** graph has many edges.
 - Dense graphs will have many matrix entries of 1.
- Determining if two nodes are adjacent is $O(1)$
- The size of the matrix is **independent** of the number of edges.
- An adjacency matrix may be a more desirable representation for dense graphs.

Adjacency Matrix

- Space required is $O(N^2)$
 - A sparse graph has few edges
 - Sparse graphs will have many matrix entries of 0.
 - a dense graph has many edges.
 - Dense graphs will have many matrix entries of 1.
- Determining if two nodes are adjacent is $O(1)$
- The size of the matrix is **independent** of the number of edges.
- An adjacency matrix may be a more desirable representation for dense graphs.

Adjacency Matrix

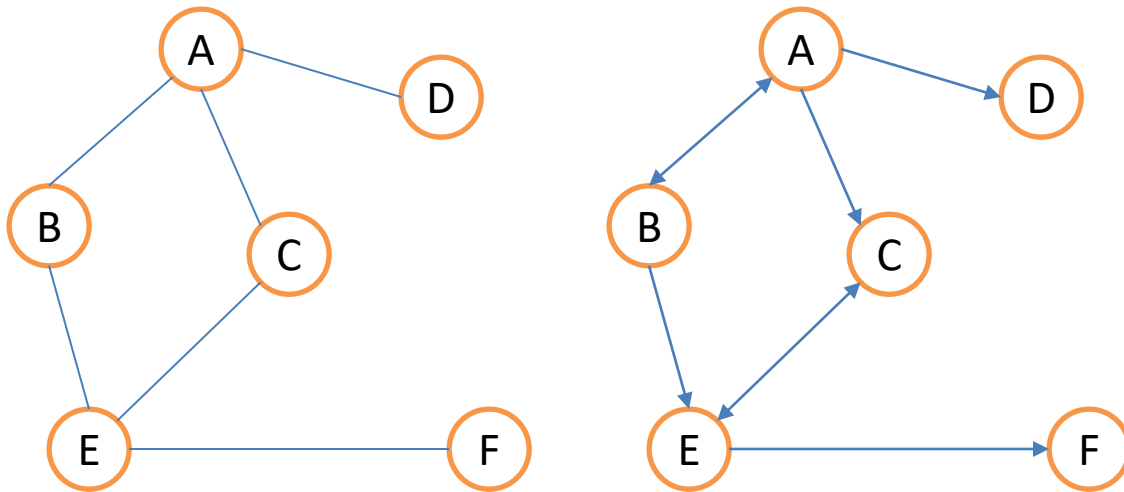
- Space required is $O(N^2)$
 - A sparse graph has few edges
 - Sparse graphs will have many matrix entries of 0.
 - a dense graph has many edges.
 - Dense graphs will have many matrix entries of 1.
- Determining if two nodes are adjacent is $O(1)$
- The size of the matrix is **independent** of the number of edges.
- An adjacency matrix may be a more desirable representation for dense graphs.

Adjacency Matrix

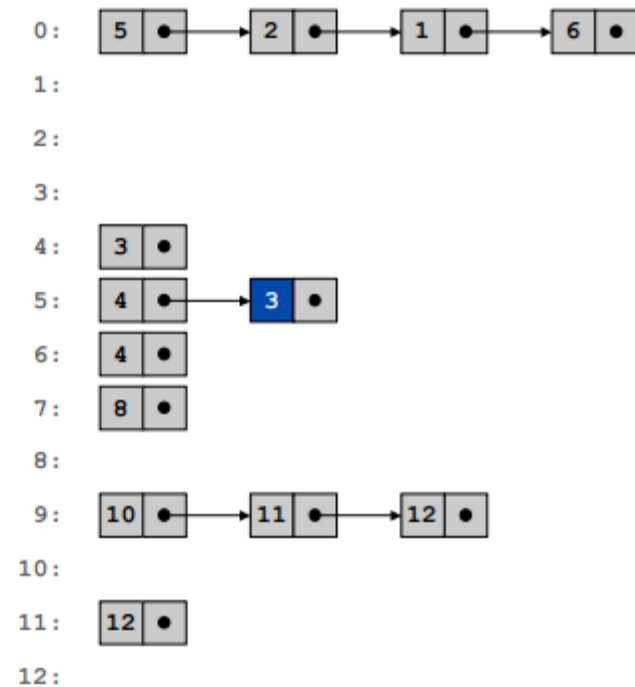
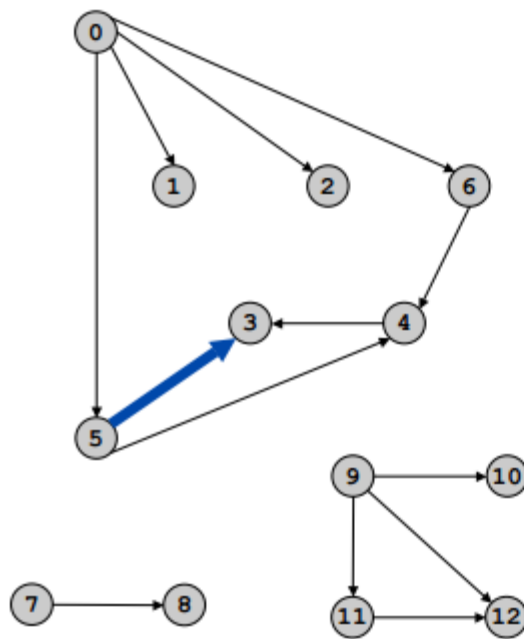
- Space required is $O(N^2)$
 - A sparse graph has few edges
 - Sparse graphs will have many matrix entries of 0.
 - a dense graph has many edges.
 - Dense graphs will have many matrix entries of 1.
- Determining if two nodes are adjacent is $O(1)$
- The size of the matrix is **independent** of the number of edges.
- An adjacency matrix may be a more desirable representation for dense graphs.

Adjacency Lists

- A graph G with N nodes represented by an array of N linked lists.
- For each x and y , if xGy is TRUE, y is on x 's list.



Adjacency Lists - Example



Adjacency Lists

- Space required is $O(N^2)$
- Density affects the lists:
 - Sparse graphs will have shorter lists.
 - Dense graphs will have longer lists.
- The order of the nodes in a list may be arbitrary.
 - A weighted graph may order them by weight
- Determining if two nodes are adjacent is $O(N)$ in the worst case. Could be much less if there are few edges.
- The number of nodes in the lists is **dependent** on the number of edges.
- An adjacency list may be a more desirable representation for sparse graphs.

Adjacency Lists

- Space required is $O(N^2)$
- Density affects the lists:
 - Sparse graphs will have shorter lists.
 - Dense graphs will have longer lists.
- The order of the nodes in a list may be arbitrary.
 - A weighted graph may order them by weight
- Determining if two nodes are adjacent is $O(N)$ in the worst case. Could be much less if there are few edges.
- The number of nodes in the lists is **dependent** on the number of edges.
- An adjacency list may be a more desirable representation for sparse graphs.

Adjacency Lists

- Space required is $O(N^2)$
- Density affects the lists:
 - Sparse graphs will have shorter lists.
 - Dense graphs will have longer lists.
- The order of the nodes in a list may be arbitrary.
 - A weighted graph may order them by weight
- Determining if two nodes are adjacent is $O(N)$ in the worst case. Could be much less if there are few edges.
- The number of nodes in the lists is **dependent** on the number of edges.
- An adjacency list may be a more desirable representation for sparse graphs.

Adjacency Lists

- Space required is $O(N^2)$
- Density affects the lists:
 - Sparse graphs will have shorter lists.
 - Dense graphs will have longer lists.
- The order of the nodes in a list may be arbitrary.
 - A weighted graph may order them by weight
- Determining if two nodes are adjacent is $O(N)$ in the worst case. Could be much less if there are few edges.
- The number of nodes in the lists is **dependent** on the number of edges.
- An adjacency list may be a more desirable representation for sparse graphs.

Adjacency Lists

- Space required is $O(N^2)$
- Density affects the lists:
 - Sparse graphs will have shorter lists.
 - Dense graphs will have longer lists.
- The order of the nodes in a list may be arbitrary.
 - A weighted graph may order them by weight
- Determining if two nodes are adjacent is $O(N)$ in the worst case. Could be much less if there are few edges.
- The number of nodes in the lists is **dependent** on the number of edges.
- An adjacency list may be a more desirable representation for sparse graphs.

Adjacency Lists

- Space required is $O(N^2)$
- Density affects the lists:
 - Sparse graphs will have shorter lists.
 - Dense graphs will have longer lists.
- The order of the nodes in a list may be arbitrary.
 - A weighted graph may order them by weight
- Determining if two nodes are adjacent is $O(N)$ in the worst case. Could be much less if there are few edges.
- The number of nodes in the lists is **dependent** on the number of edges.
- An adjacency list may be a more desirable representation for sparse graphs.

Graph Traversals

Graph Traversals

- Unlike tree traversals, there is no "starting" (i.e. root) node in a graph.
- Choosing an arbitrary starting node will not guarantee that all nodes are visited.
- The search must systematically traverse all of the edges in order to discover all of the vertices.
- Although it sounds like a lot of redundant work, it can be accomplished in $O(N)$ time, where N is the number of vertices.

Graph Traversals

- Unlike tree traversals, there is no "starting" (i.e. root) node in a graph.
- Choosing an arbitrary starting node will not guarantee that all nodes are visited.
- The search must systematically traverse all of the edges in order to discover all of the vertices.
- Although it sounds like a lot of redundant work, it can be accomplished in $O(N)$ time, where N is the number of vertices.

Graph Traversals

- Unlike tree traversals, there is no "starting" (i.e. root) node in a graph.
- Choosing an arbitrary starting node will not guarantee that all nodes are visited.
- The search must systematically traverse all of the edges in order to discover all of the vertices.
- Although it sounds like a lot of redundant work, it can be accomplished in $O(N)$ time, where N is the number of vertices.

Graph Traversals

- Unlike tree traversals, there is no "starting" (i.e. root) node in a graph.
- Choosing an arbitrary starting node will not guarantee that all nodes are visited.
- The search must systematically traverse all of the edges in order to discover all of the vertices.
- Although it sounds like a lot of redundant work, it can be accomplished in $O(N)$ time, where N is the number of vertices.

Graph Traversals

- Breath-first traversal
- Depth-first traversal

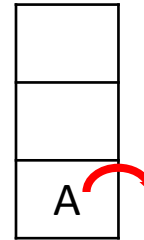
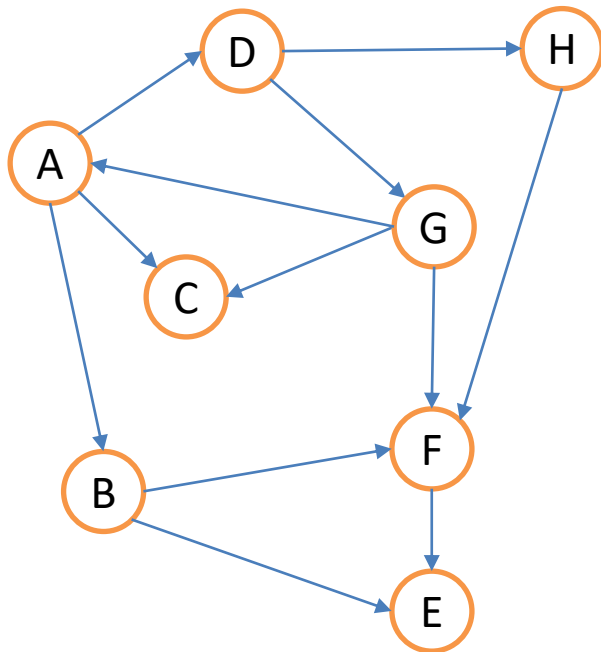
Pseudo-code for Graphs Traversals

```
GraphSearch(G is the graph to search, v is the starting vertex){  
    Put v into container C;  
    while (container C is not empty){  
        Remove a vertex, x, from container C;  
        if (x has not been visited){  
            Visit x;  
            Set x.visited to TRUE;  
            for (each vertex, w, adjacent to x){  
                if (w has not been visited)  
                    Put w into container C;  
            }//end for  
        }//end if  
    }//end while  
} //end GraphSearch
```

Graph Traversals: Depth First

Container: Stack

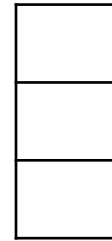
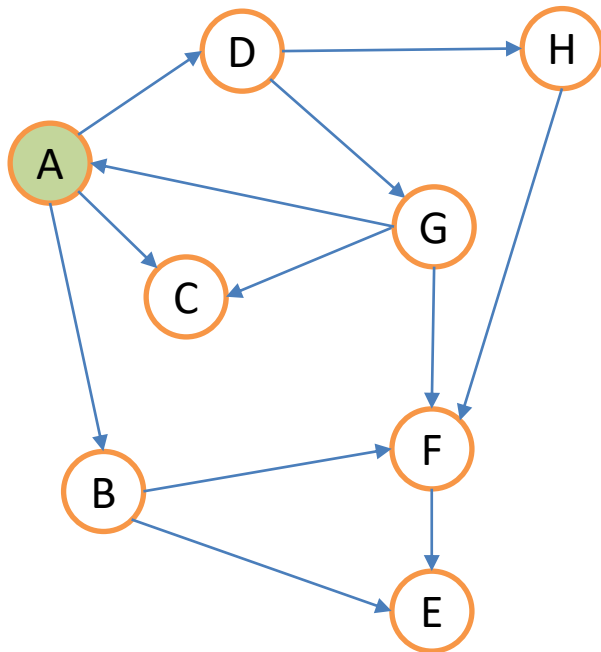
Start at A



Graph Traversals: Depth First

Container: Stack

Start at A



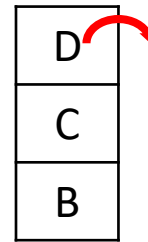
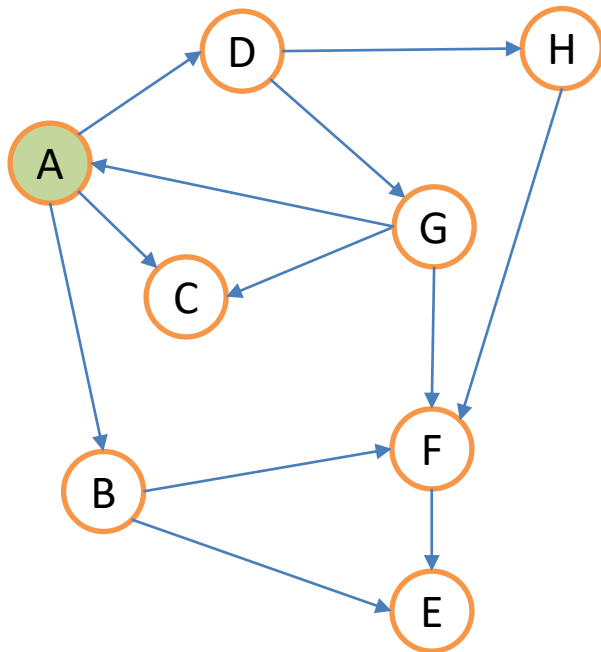
Traversal Order

A

Graph Traversals: Depth First

Container: Stack

Start at A



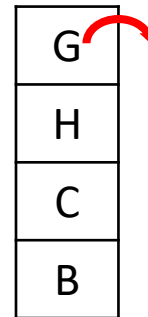
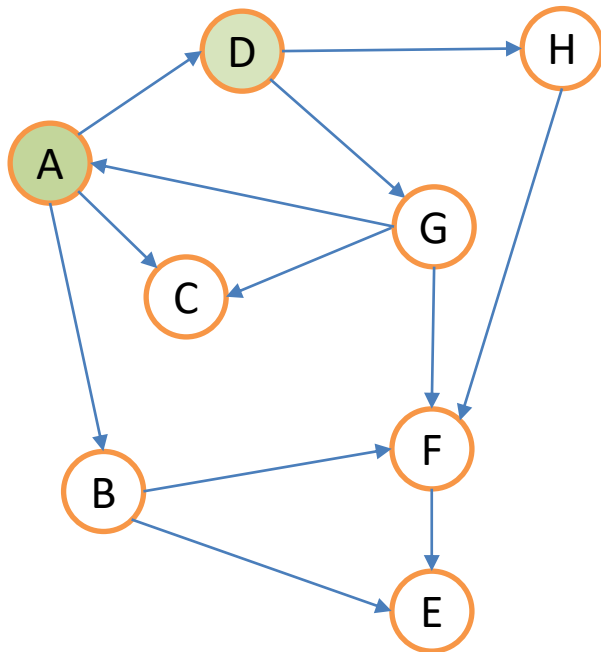
Traversal Order

A

Graph Traversals: Depth First

Container: Stack

Start at A



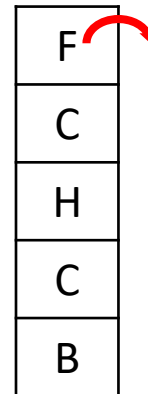
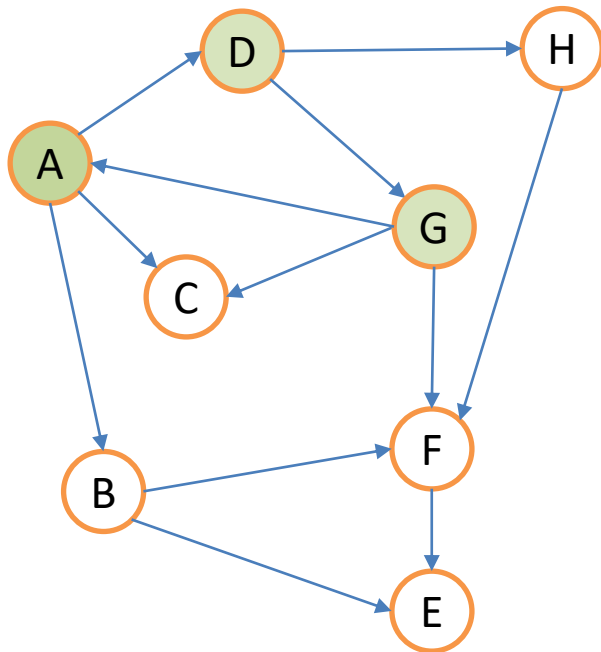
Traversal Order

AD

Graph Traversals: Depth First

Container: Stack

Start at A

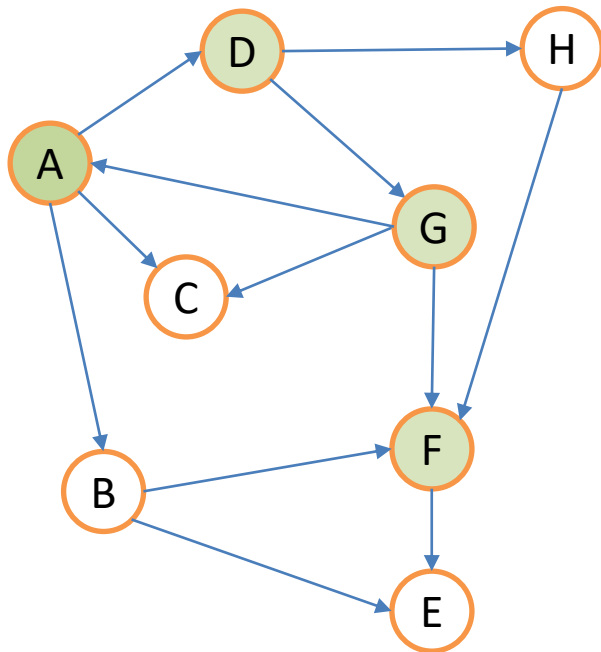


Traversal Order
ADG

Graph Traversals: Depth First

Container: Stack

Start at A



E
C
H
C
B

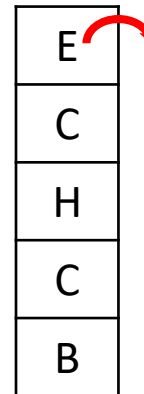
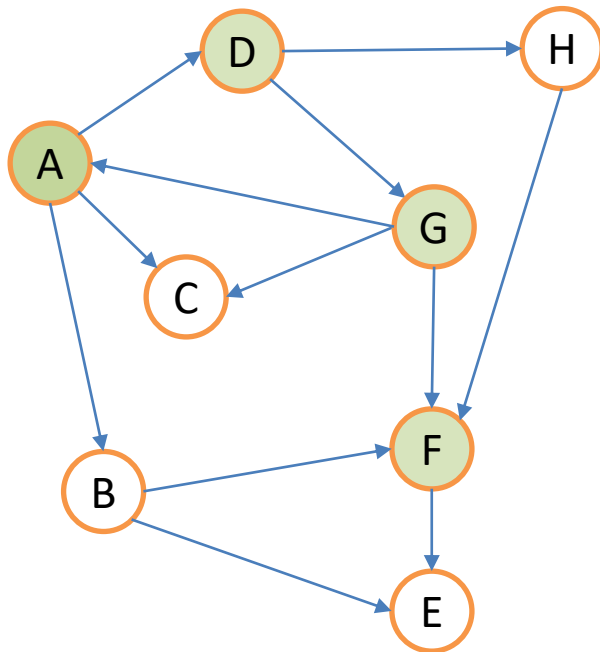
Traversal Order

ADGF

Graph Traversals: Depth First

Container: Stack

Start at A

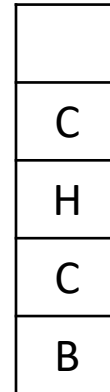
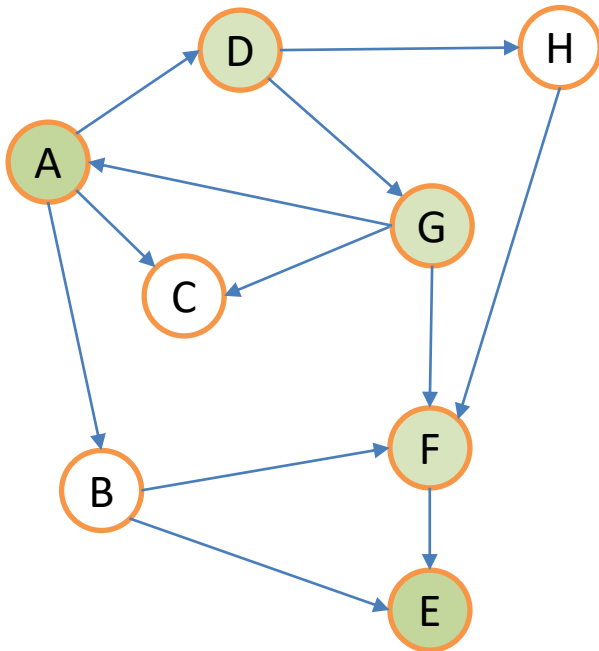


Traversal Order
ADGF

Graph Traversals: Depth First

Container: Stack

Start at A



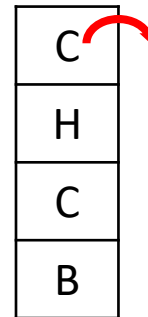
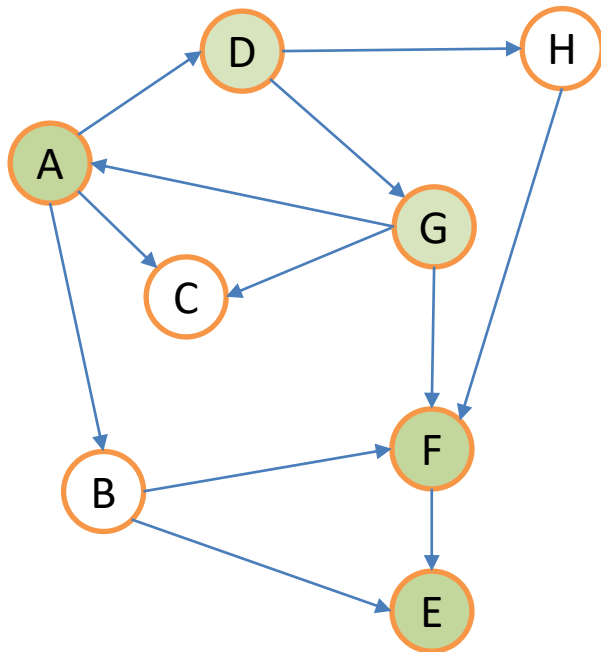
Traversal Order

ADGFE

Graph Traversals: Depth First

Container: Stack

Start at A

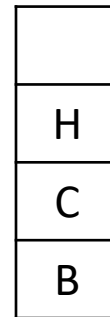
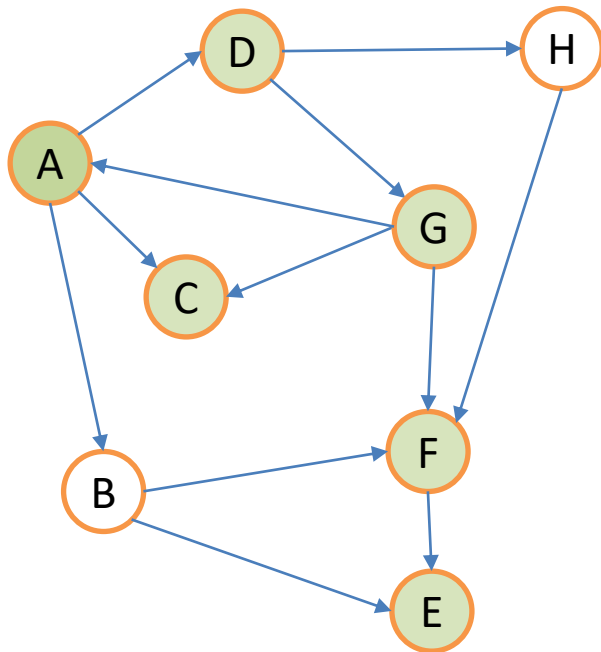


Traversal Order
ADGFE

Graph Traversals: Depth First

Container: Stack

Start at A

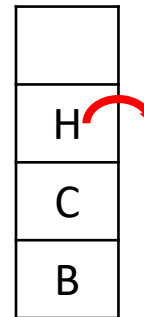
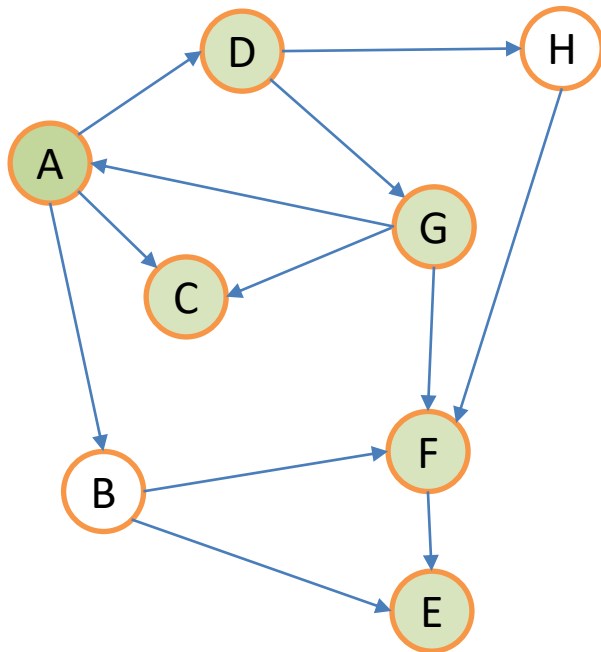


Traversal Order
ADGFEC

Graph Traversals: Depth First

Container: Stack

Start at A

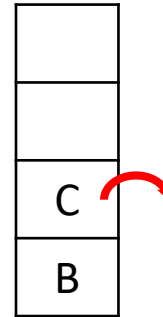
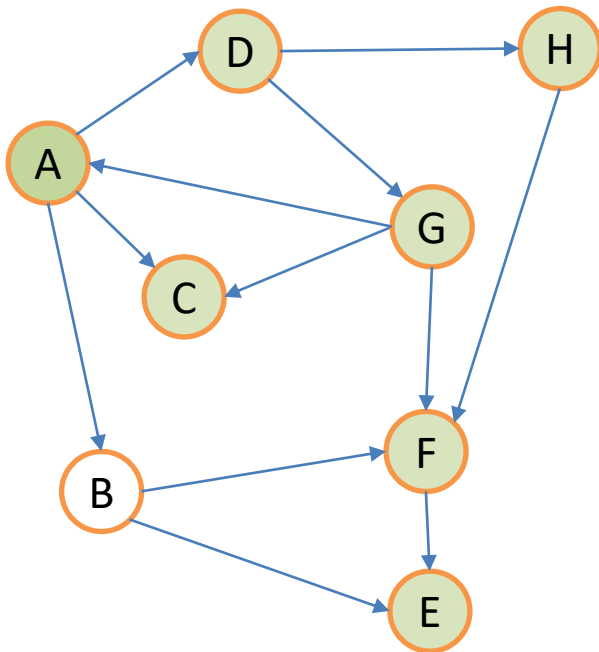


Traversal Order
ADGFEC

Graph Traversals: Depth First

Container: Stack

Start at A

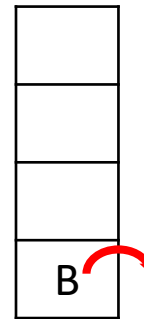
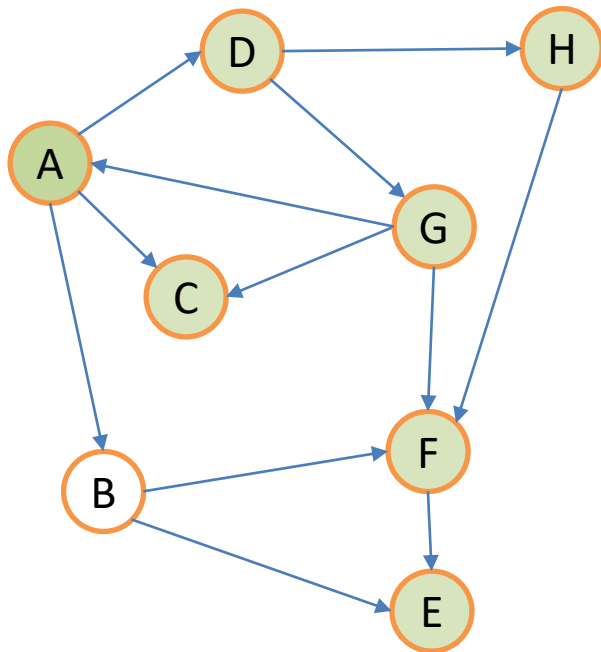


Traversal Order
ADGFECH

Graph Traversals: Depth First

Container: Stack

Start at A

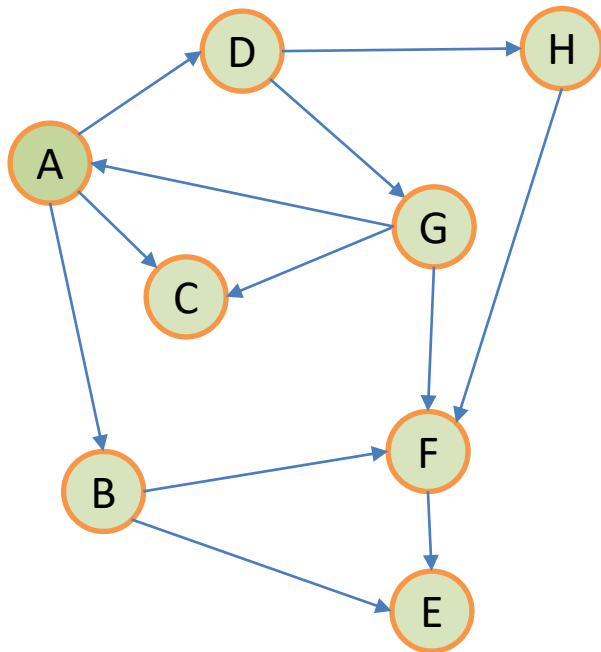


Traversal Order
ADGFECH

Graph Traversals: Depth First

Container: Stack

Start at A

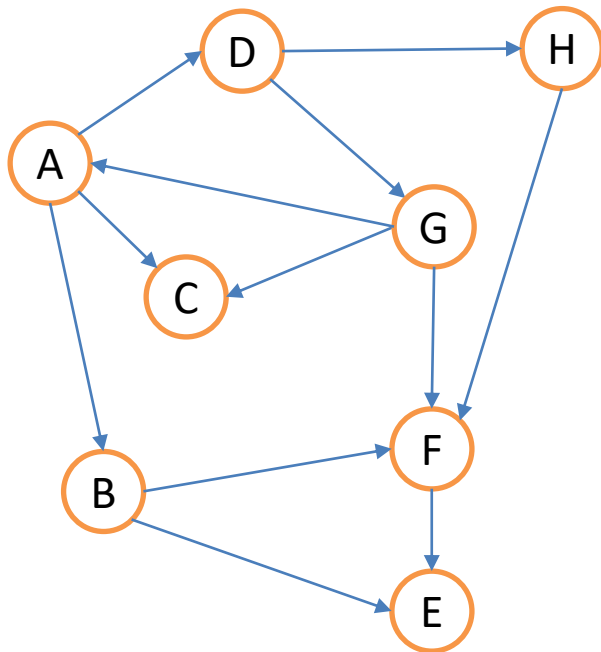
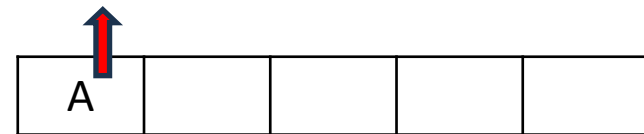


Traversal Order
ADGFECHB

Graph Traversals: Breadth First

Container: Queue

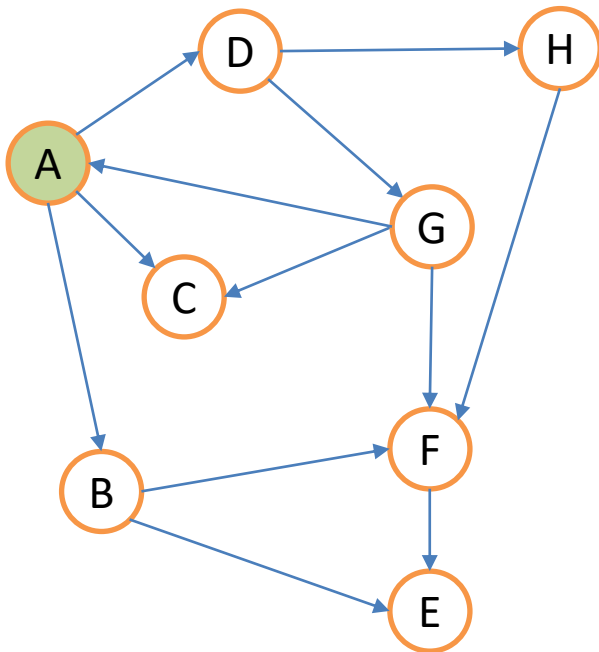
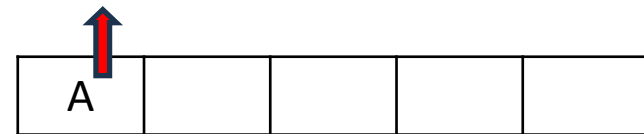
Start at A



Graph Traversals: Breadth First

Container: Queue

Start at A



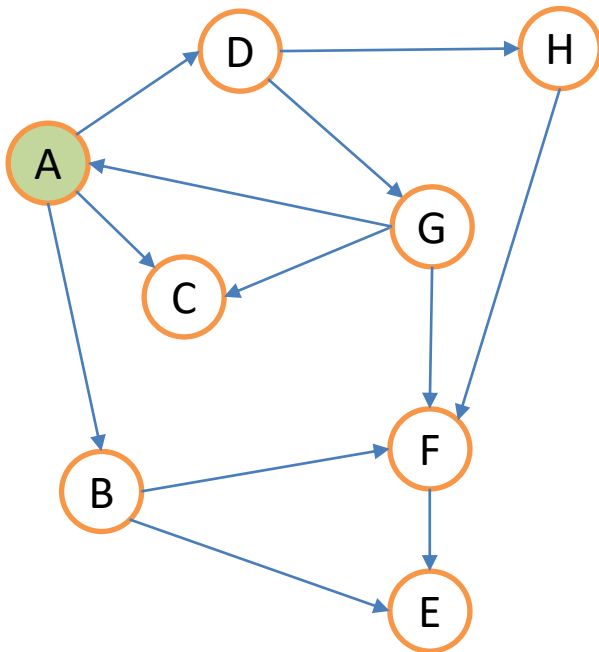
Traversal Order

A

Graph Traversals: Breadth First

Container: Queue

Start at A



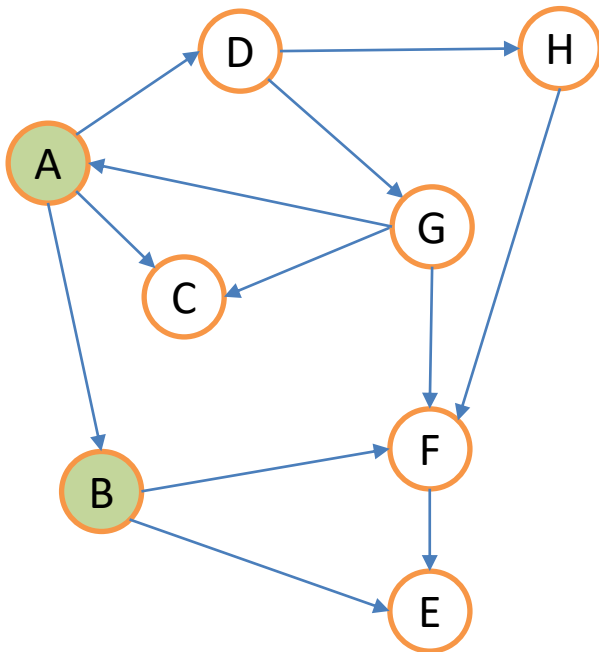
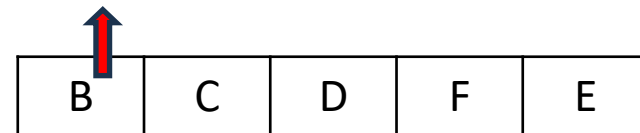
Traversal Order

A

Graph Traversals: Breadth First

Container: Queue

Start at A



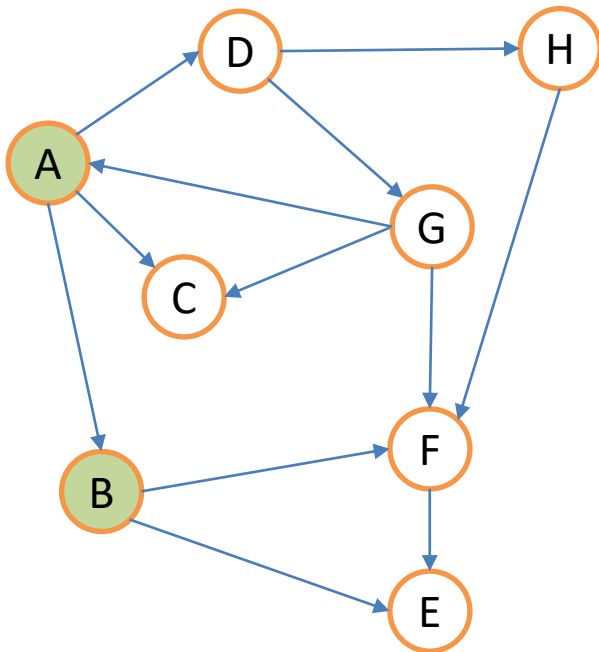
Traversal Order

A B

Graph Traversals: Breadth First

Container: Queue

Start at A



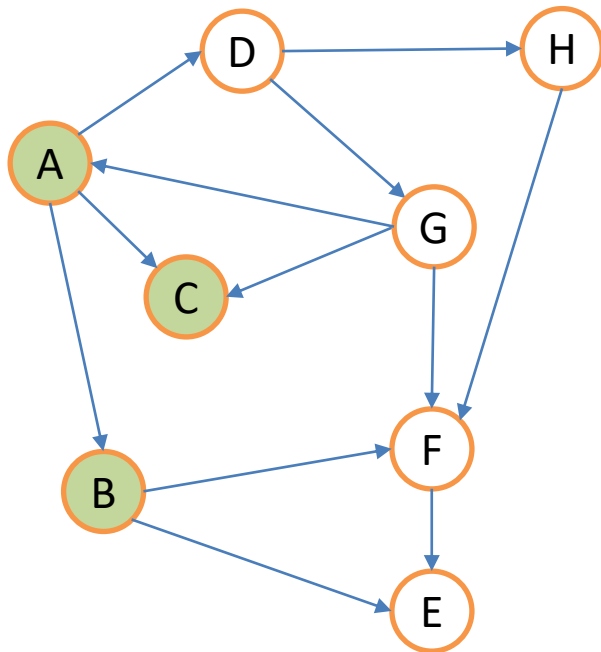
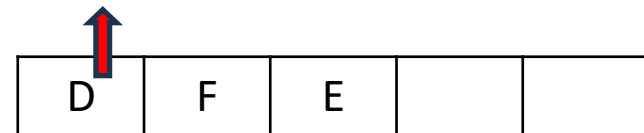
Traversal Order

A B

Graph Traversals: Breadth First

Container: Queue

Start at A



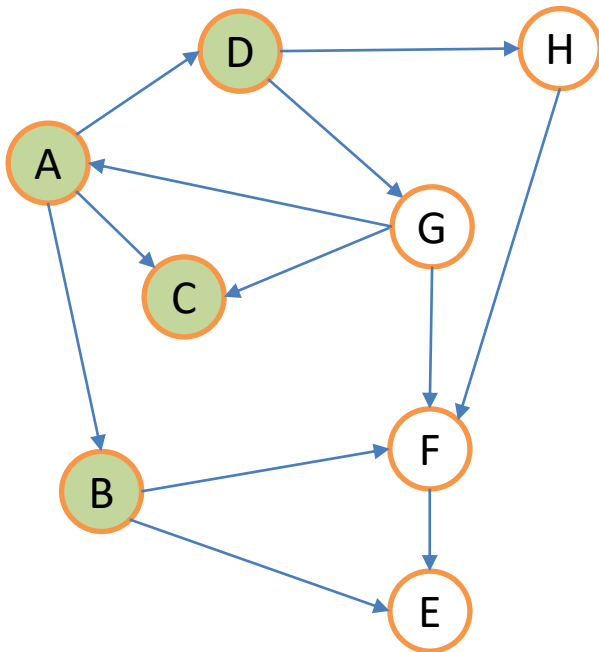
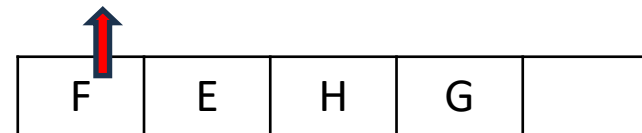
Traversal Order

A B C

Graph Traversals: Breadth First

Container: Queue

Start at A



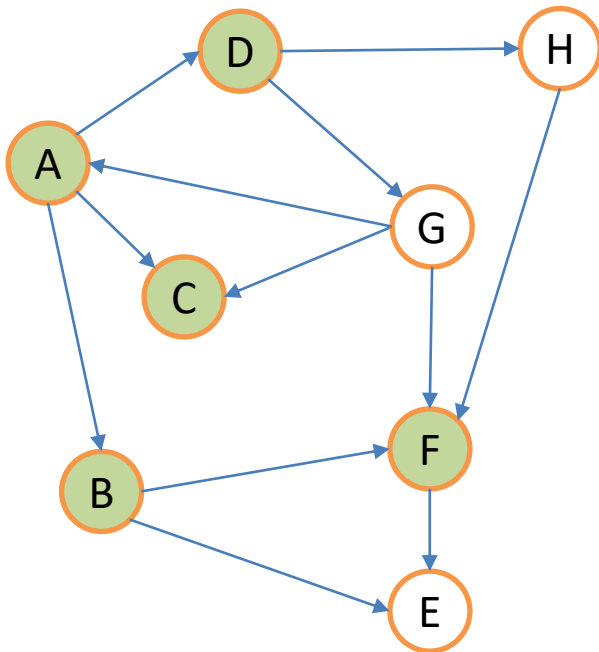
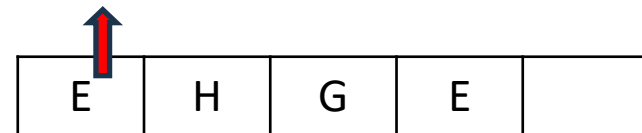
Traversal Order

A B C D

Graph Traversals: Breadth First

Container: Queue

Start at A



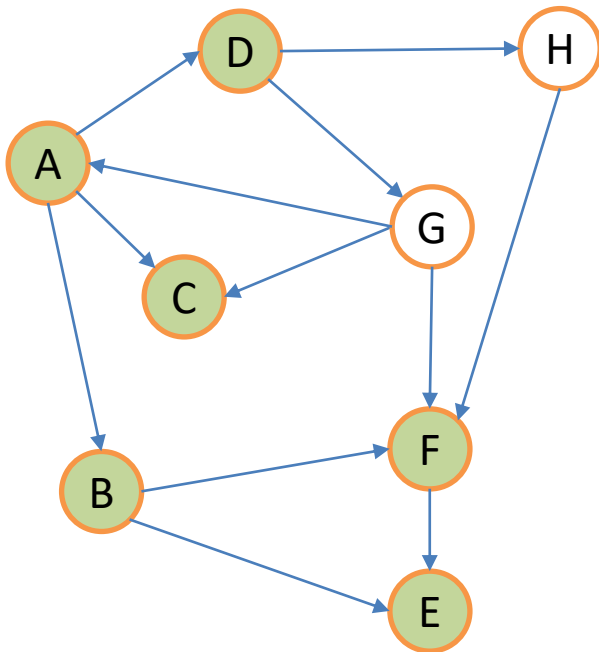
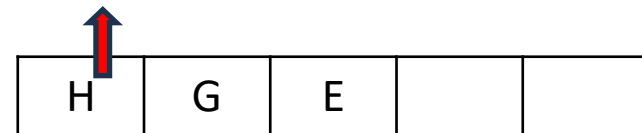
Traversal Order

A B C D F

Graph Traversals: Breadth First

Container: Queue

Start at A



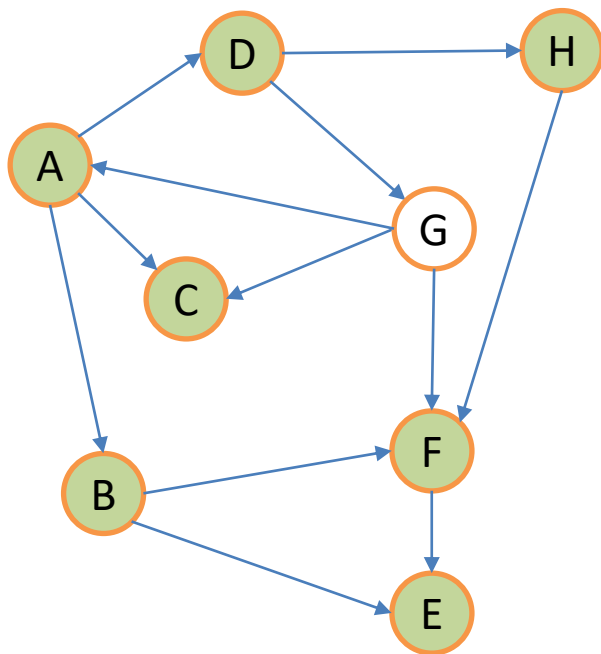
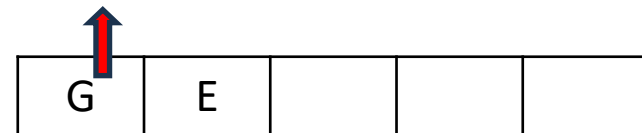
Traversal Order

A B C D F E

Graph Traversals: Breadth First

Container: Queue

Start at A



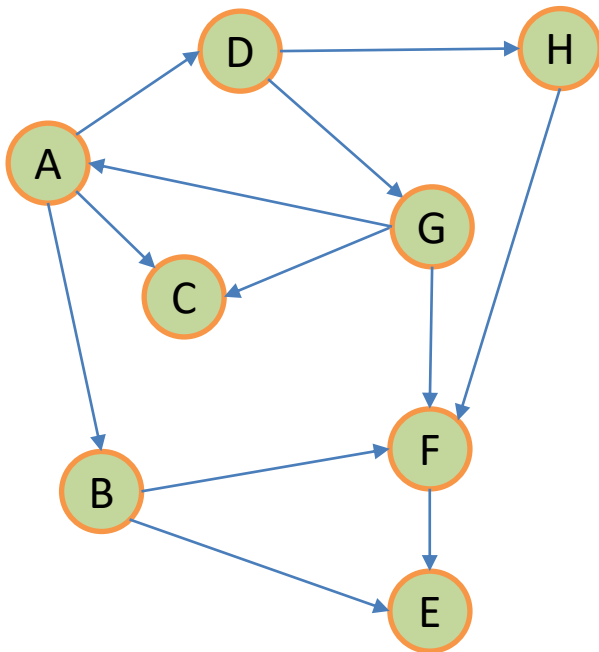
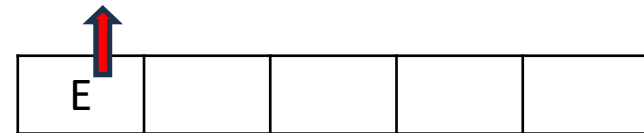
Traversal Order

A B C D F E H

Graph Traversals: Breadth First

Container: Queue

Start at A



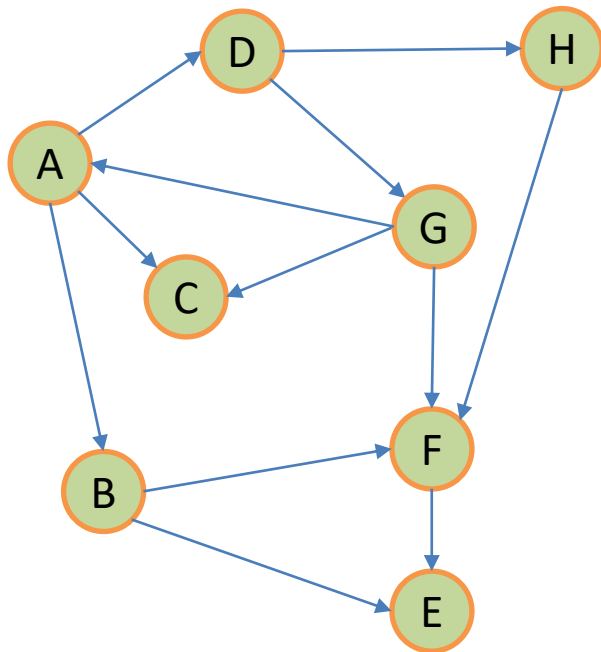
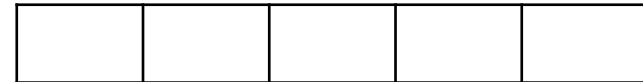
Traversal Order

A B C D F E H G

Graph Traversals: Breadth First

Container: Queue

Start at A



Traversal Order

A B C D F E H G

Graph Traversals

- Breath-first traversal
- Depth-first traversal
- **Example 1: Starting at A**
 - If *C is a Stack*, one order of traversal is:
 - *A, D, H, F, E, G, C, B*
 - Another traversal is: A, B, E, F, C, D, G, H
 - If *C is a Queue*, one order of traversal is:
 - *A, B, C, D, E, F, G, H*
 - Another traversal is: A, D, C, B, H, G, F, E

Notes

- Depth-first: Descendants are visited before siblings.
 - To traverse depth-first, use a **Stack**.
- Breadth-first: siblings are visited before descendants.
 - To traverse breadth-first, use a **Queue**.
- For all vertices to be visited from any node, the graph must be **strongly connected**.
- For **weakly connected** graphs, you'd need to exhaustively traverse from every vertex.

```
for each vertex, v, in graph, G  
    GraphSearch(G, v)
```