Topic 10 - Graphs
Lecture 10a - Introduction

CSCI 240

Data Structures and Algorithms

Prof. Dominick Atanasio

Today

Agenda

- Graph Terminology
- Graph Representations
- Graph Traversals

Graph Terminology

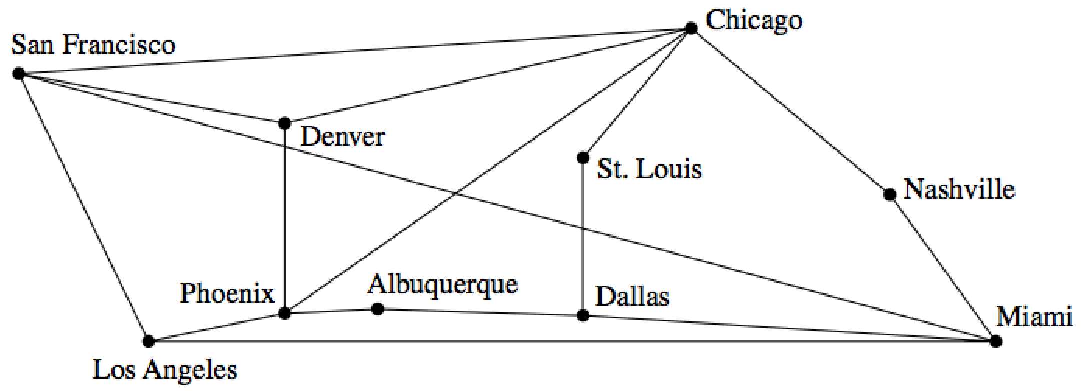
- Graphs are the most general data structure. They are also commonly used data structures.
- Graph is a non-linear data structure consisting of nodes and edges between nodes.
- G = (V, E) where V is a set of vertices (nodes) and E is a set of edges

Graph Terminology

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

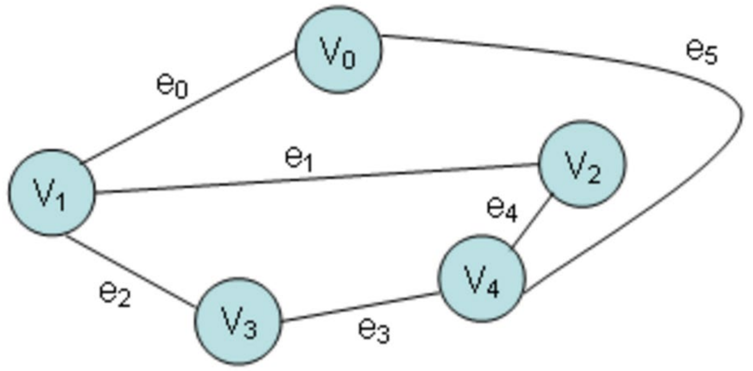
- Graph is a non-linear data structure consisting of nodes and links between nodes.

  - 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**.

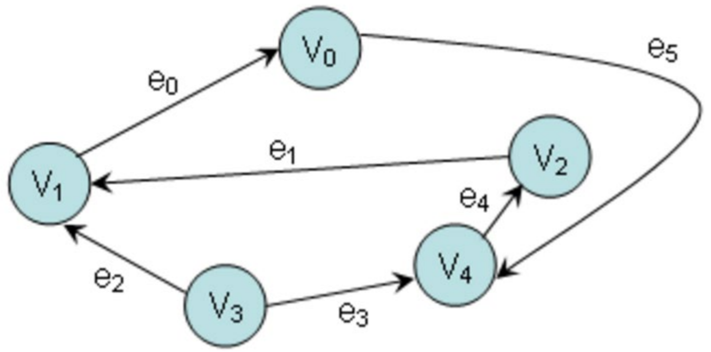


Graph Terminology

- Types of graphs
  - **Undirected graph**
  - **Directed graph**



Undirected graph

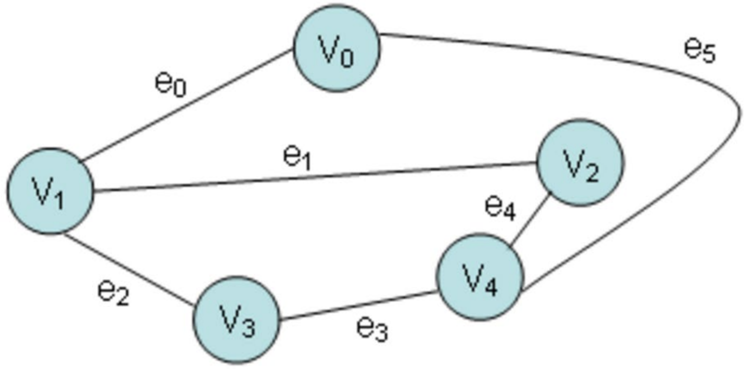


Directed graph

- Graphs in Problem Solving:
  - Often a problem can be represented as a graph, and the solution to the problem is obtained by solving a problem on the corresponding graph.

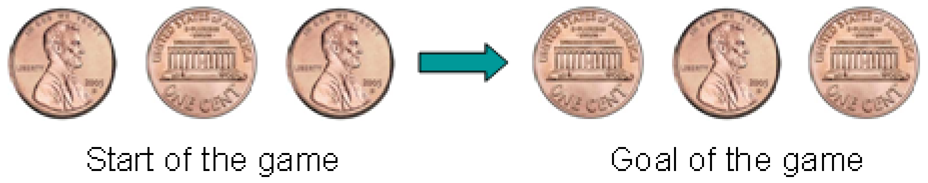
Graph Terminology

- **Undirected graph**
  - An undirected graph is a set of nodes and a set of links between the nodes.
  - Each node is called a **vertex**, each link is called an **edge**, and each edge connects two vertices.
  - The order of the two connected vertices is unimportant.
  - An undirected graph is a finite set of vertices together with a finite set of edges. Both sets might be empty, which is called the empty graph.



Graph Terminology

- Example: Coin Game (Undirected State Graphs)

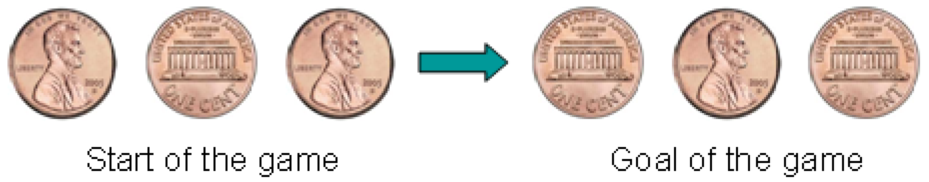


- Rules:
  - You may flip the middle coin whenever you want to.
  - You may flip one of the end coins only if the other two coins are the same as each other.

- The possible resulting states can be modelled with a graph
  - Each state as a vertex
  - Each transition from one state to the next as an edge

Graph Terminology

- Example: Coin Game (Undirected State Graphs)

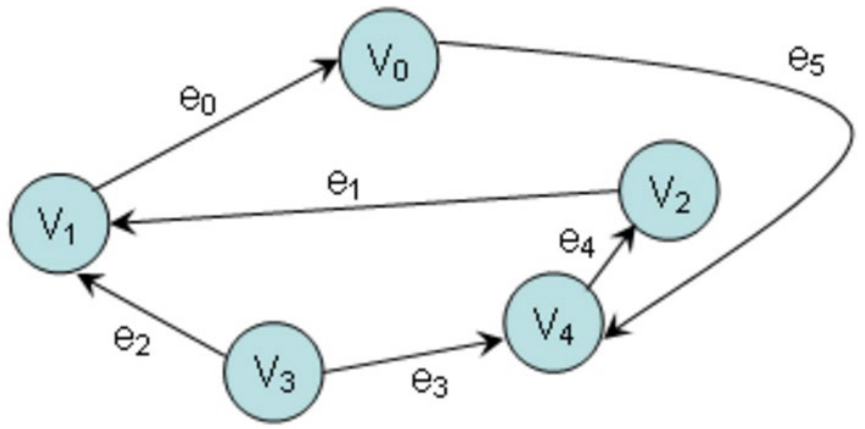


- Rules:
  - You may flip the middle coin whenever you want to.
  - You may flip one of the end coins only if the other two coins are the same as each other.

To solve the above problem, we are going to build an undirected state graph. Once the undirected state graph is created, the game becomes a problem of finding a path from one vertex to another, where the path is allowed only to follow edges.

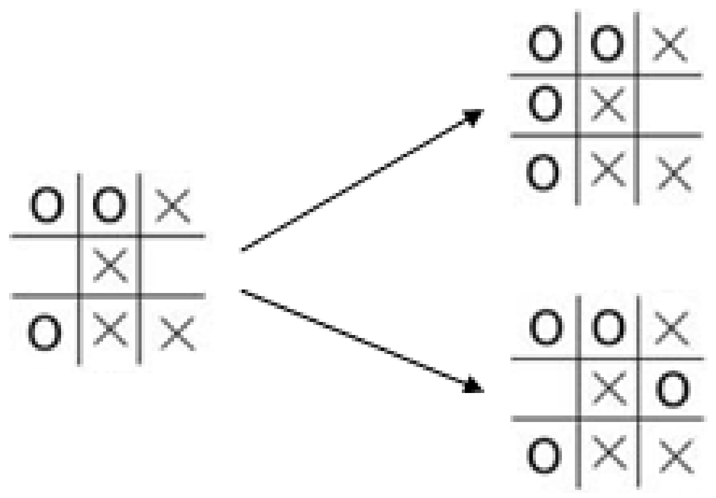
Graph Terminology

- Directed graph
  - Each **edge** is associated with two vertices, called its source and target vertices.
  - We say that the edge connects its source to its target.
  - The order of the two connected vertices is important.
  - A directed graph is a **finite set of vertices** together with a **finite collection of edges**. Both might be empty, which is called the empty graph.



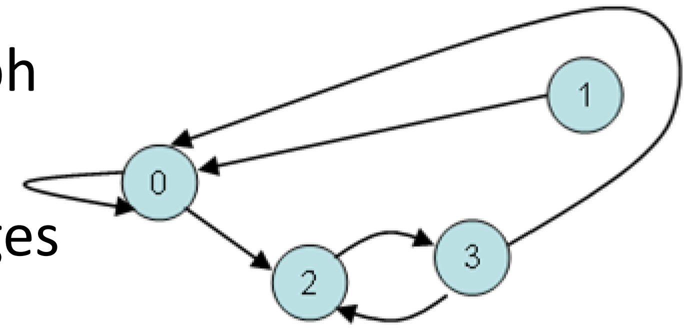
Graph Terminology

- Example: tic-tac-toe (Directed State Graphs)
  - Reversing a move is sometimes forbidden



Graph Terminology

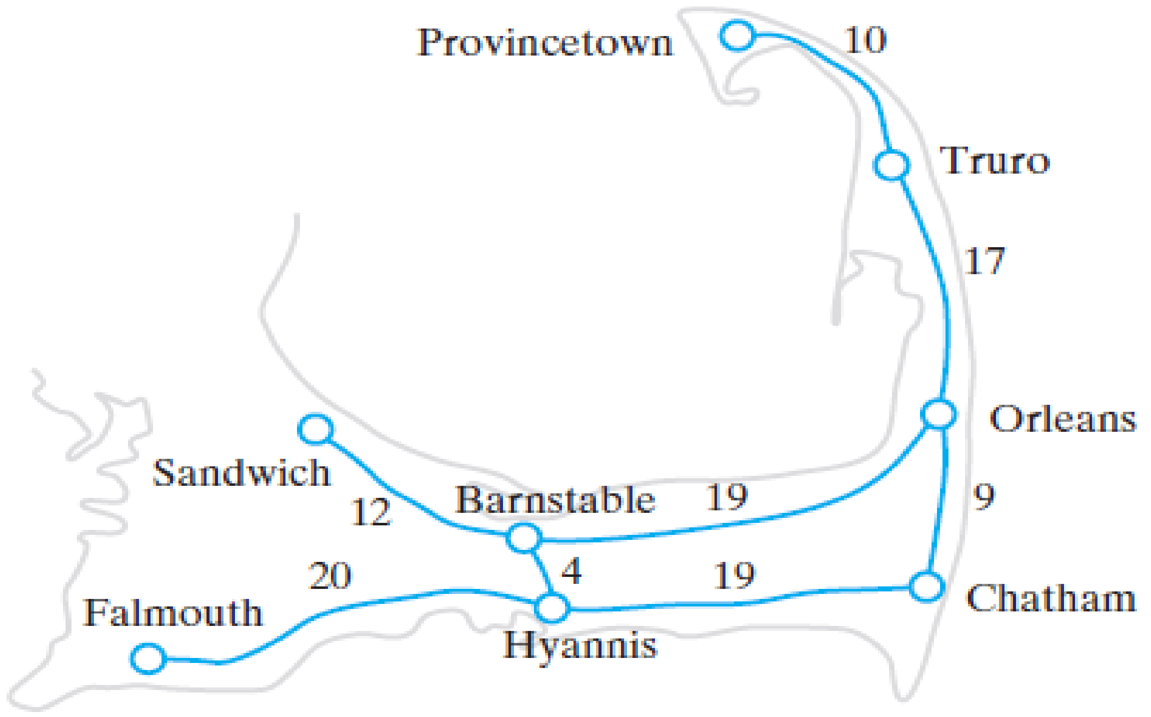
- **Loop**: an edge that connects a vertex to itself.
- **Path**: a sequence of vertices, p0, p1, ..., pm, such that each adjacent pair of vertices pi and pi+1 are connected by an edge.
- **Cycle**: a simple path with no repeated vertices or edges other than the starting and ending vertices. A cycle in a directed graph is called a directed cycle.
- **Multiple edges**: in principle, a graph can have two or more edges connecting the same two vertices in the same direction.
- **Simple graphs**: the graphs that have no loops and no multiple edges. In fact, many applications require only simple directed graphs or even simple undirected graphs.



A tree is a special type of graph without cycle (acyclic)

Graph Terminology

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



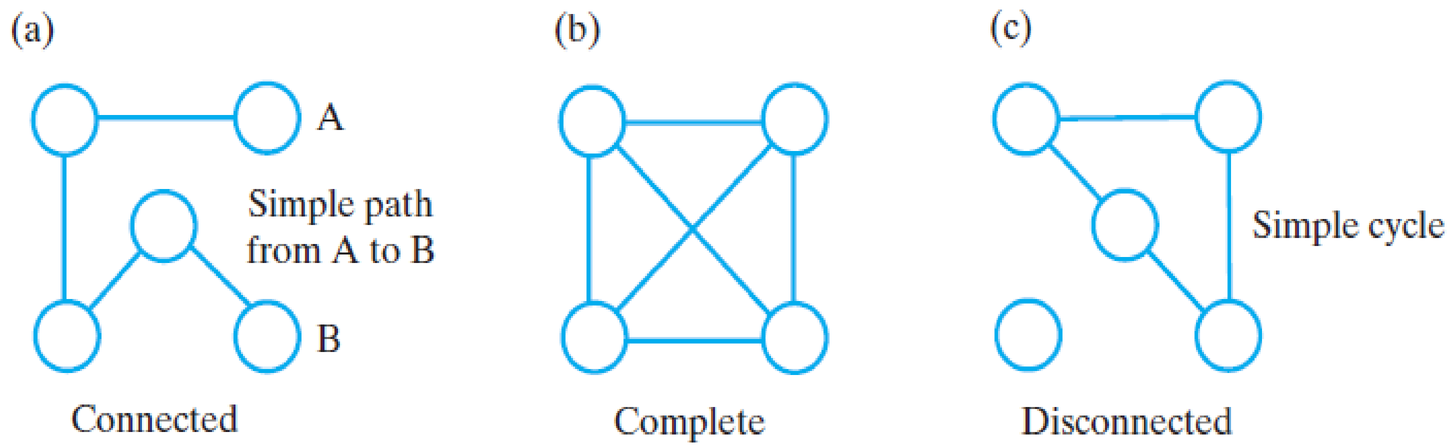
A weighted graph

Graph Terminology

- Connected graph
  - A graph that has a path between every pair of distinct vertices is connected

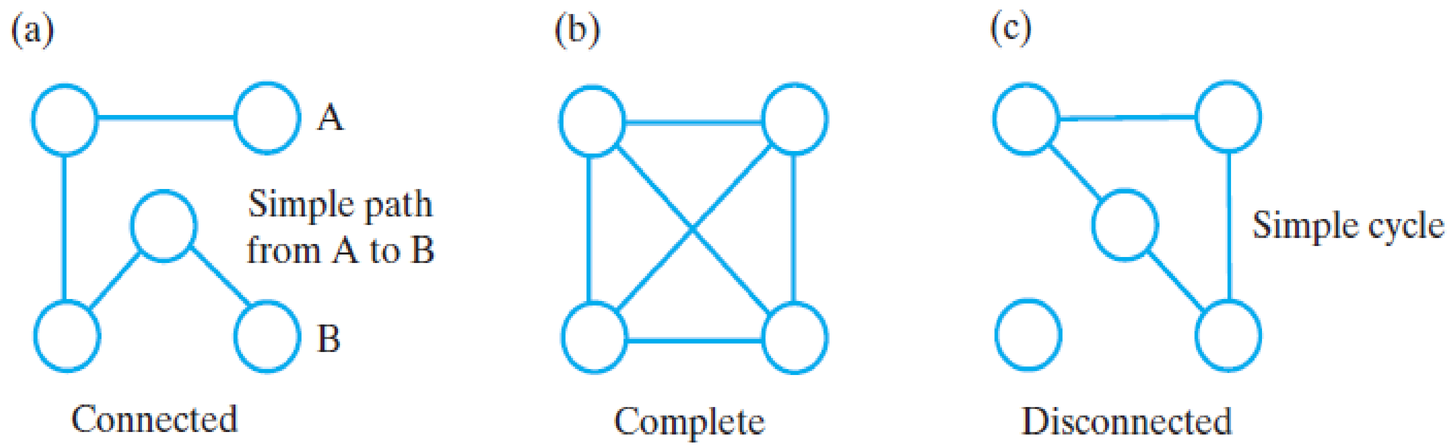
- Complete graph
  - A graph has an edge between every pair of distinct vertices.

- Undirected graphs can be
  - Connected
  - Complete or
  - Disconnected



Graph Terminology

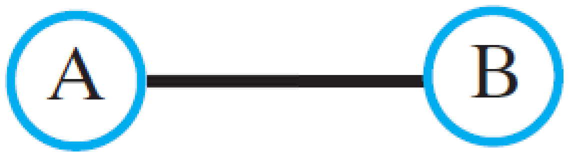
- Connected graph
  - A graph that has a path between every pair of distinct vertices is connected
- Complete graph
  - A graph has an edge between every pair of distinct vertices.
- Undirected graphs can be
  - Connected
  - Complete or
  - Disconnected



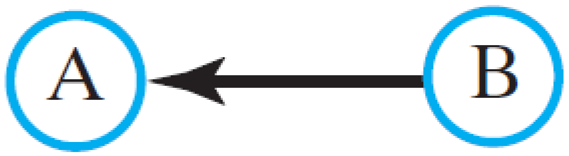
Graph Terminology

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

- Adjacent vertices are called neighbors.



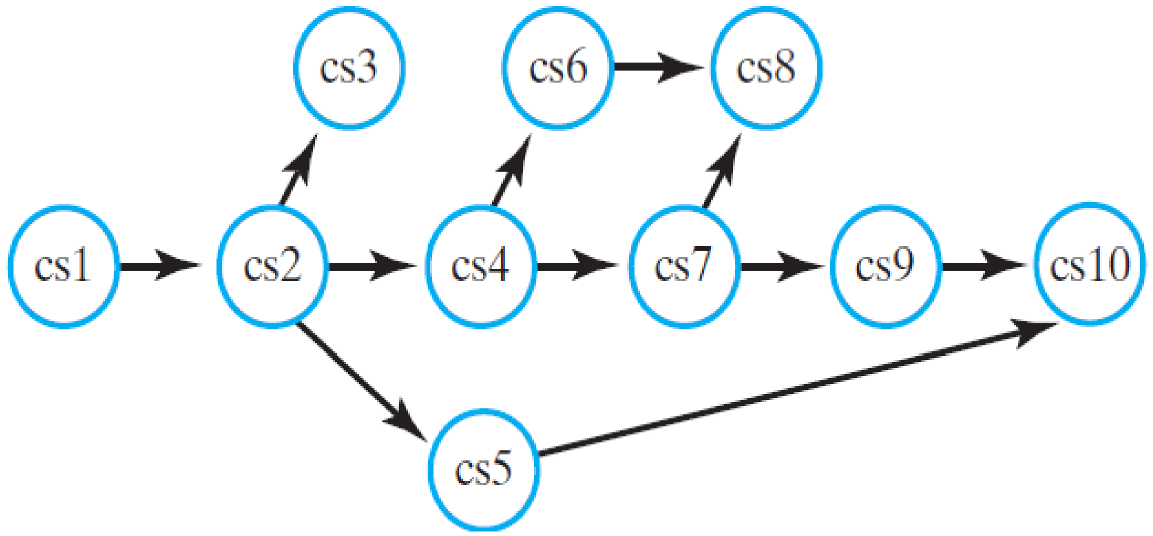
Vertex A and vertex B are adjacent



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

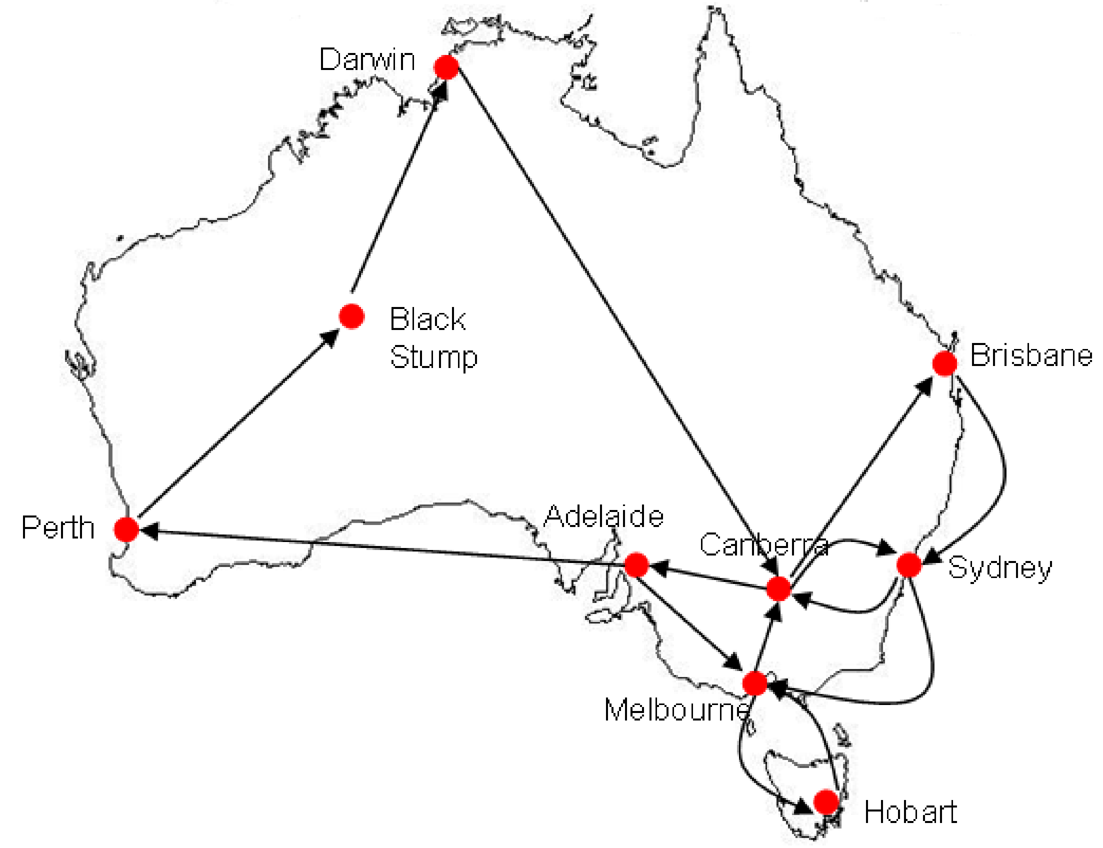
In-Class Exercise

- 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?



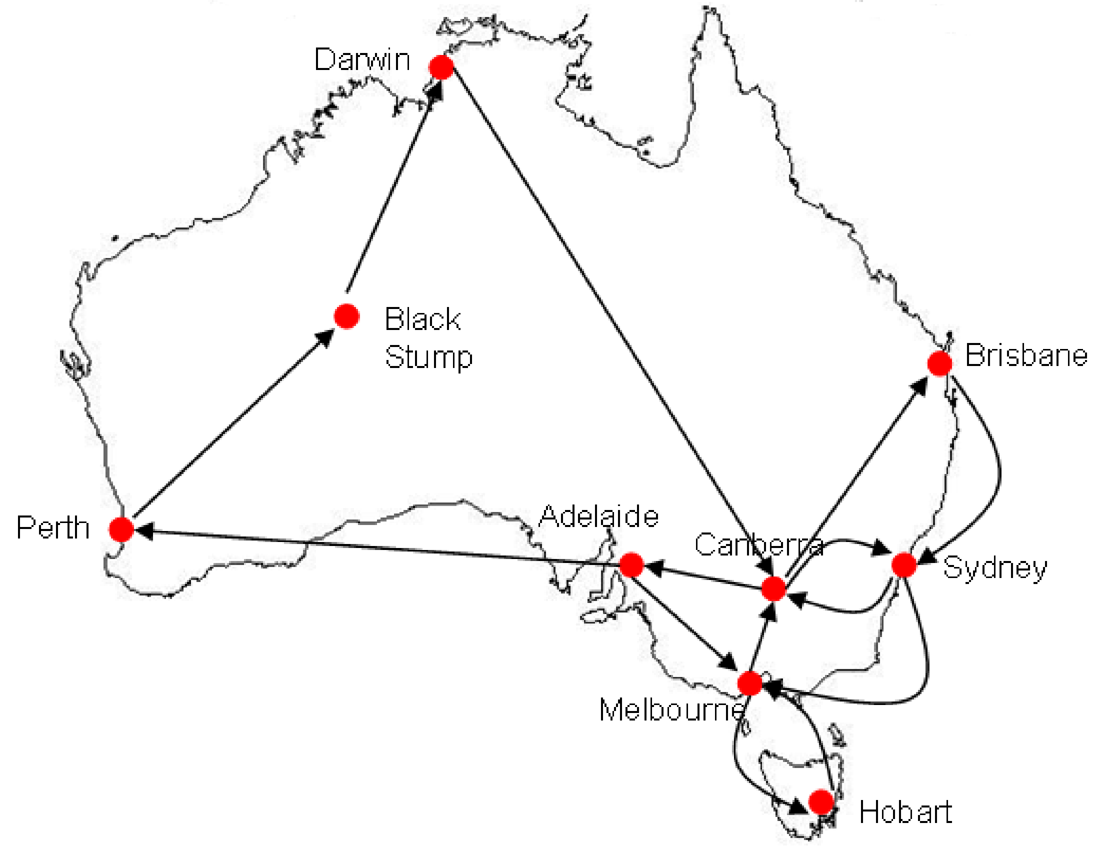
In-Class Exercise

- Airline Routing Example: Crocodile Airlines Routes
  - How many vertices and edges does the graph have? How many loops?
  - Is it a simple graph? Why or why not?
  - What is the shortest path from "Black Stump" to "Melbourne"?



In-Class Exercise

- Airline Routing Example: Crocodile Airlines Routes
  - How many vertices and edges does the graph have? How many loops?
  - Is it a simple graph? Why or why not?
  - What is the shortest path from "Black Stump" to "Melbourne"? (The shortest path problem on a graph)



In-Class Exercise

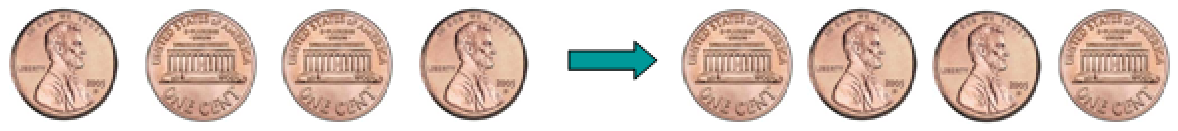
- Number of Edges in a Graph

- 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

- Suppose we have 4 coins in the coin game.



- Three rules:
  - Either of the end coins may be flipped whenever you want to.
  - A middle coin may be flipped from head to tail only if the coin to its immediate right is already heads.
  - A middle coin may be flipped from tail to head only if the coin to its immediate left is already tails.

- Draw the directed state graph for this game and determine whether it is possible to go from the start to the goal. Why does the graph need to be directed?

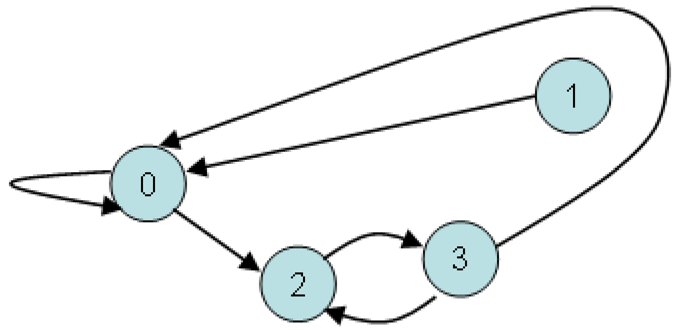
Graph Implementations

- 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

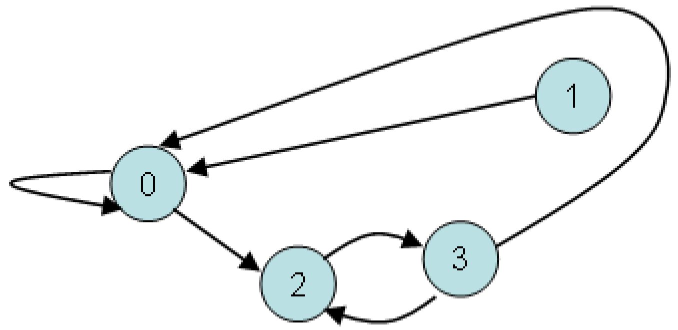
- Representing Graphs with an Adjacency Matrix
  - An adjacency matrix is a square grid of true/false values that represent the edges of a graph.
  - If the graph contains n vertices, then the grid contains n rows and n columns.
  - For two vertex numbers i and j, the component at row i and column j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



$$\begin{array}{c|cccc} & 0 & 1 & 2 & 3 \\ \hline 0 & \text{true} & \text{false} & \text{true} & \text{false} \\ 1 & \text{true} & \text{false} & \text{false} & \text{false} \\ 2 & \text{false} & \text{false} & \text{false} & \text{true} \\ 3 & \text{true} & \text{false} & \text{true} & \text{false} \end{array}$$

An Adjacency Matrix

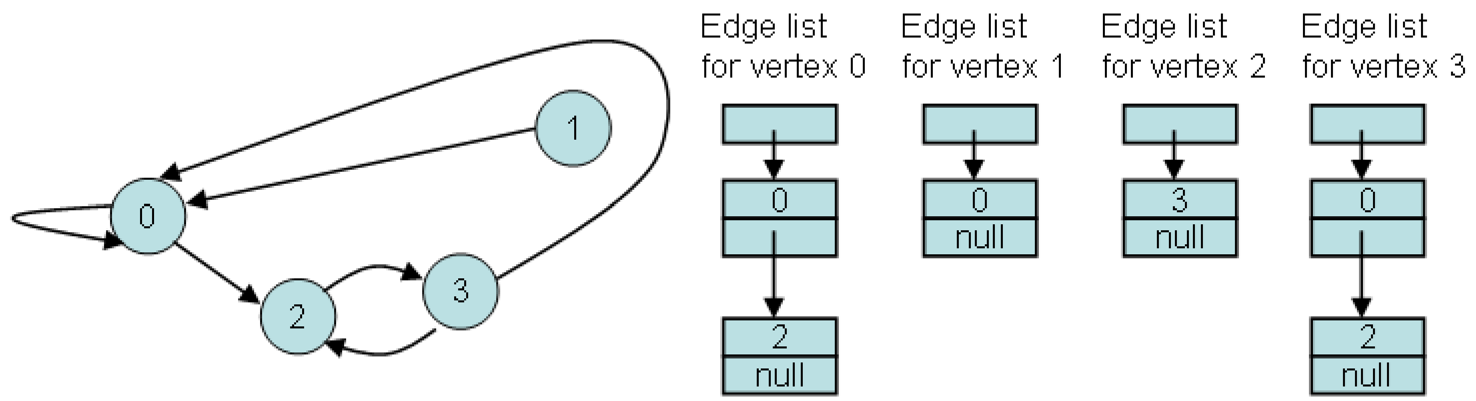
- We can use a two-dimensional array to store an adjacency matrix:
- boolean[][] adjacent = new boolean[4][4];
- Once the adjacency matrix has been set, an application can examine locations of the matrix to determine which edges are present and which are missing.



$$
\begin{array}{c}
 \\ 0 \\ 1 \\ 2 \\ 3
\end{array}
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
\left(\begin{array}{cccc}
\text{true} & \text{false} & \text{true} & \text{false} \\
\text{true} & \text{false} & \text{false} & \text{false} \\
\text{false} & \text{false} & \text{false} & \text{true} \\
\text{true} & \text{false} & \text{true} & \text{false}
\end{array}\right)
\end{array}
$$

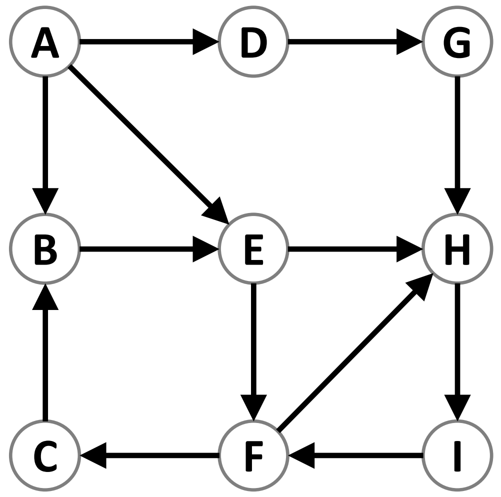
An Adjacency List

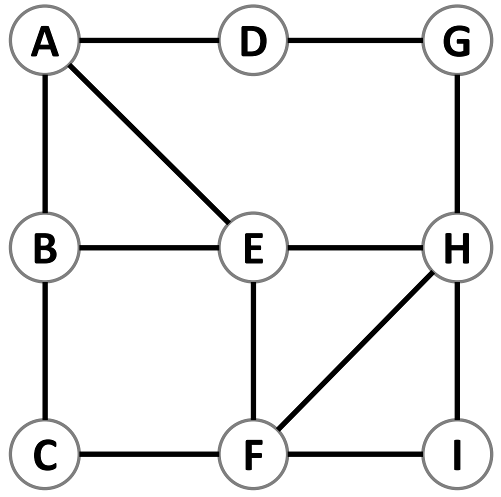
- Representing Graphs with Adjacency Lists
  - A directed graph with n vertices can be represented by n different linked lists.
  - List number i provides the connections for vertex i.
  - For each entry j in list number i, there is an edge from i to j.



In-Class Exercises

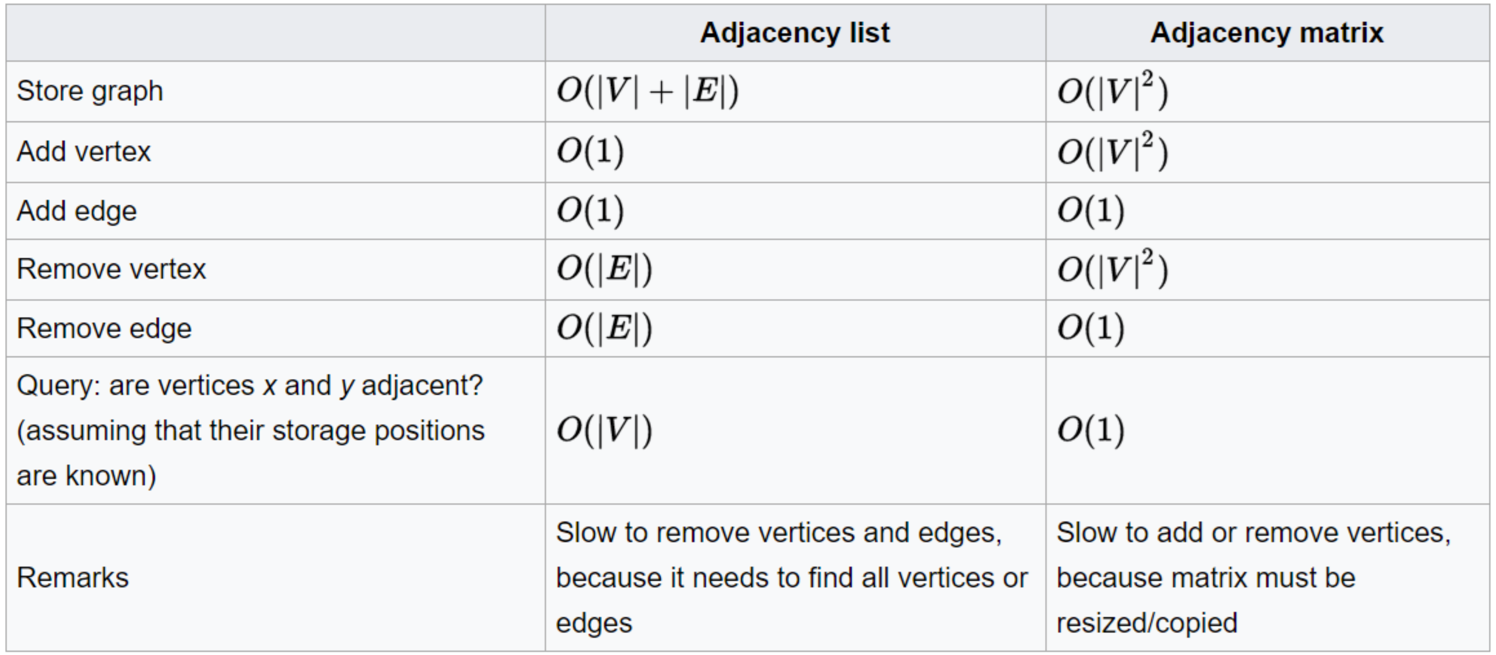
- Use adjacency matrix and adjacency list to represent the following graphs?





Which Representation is Best?

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

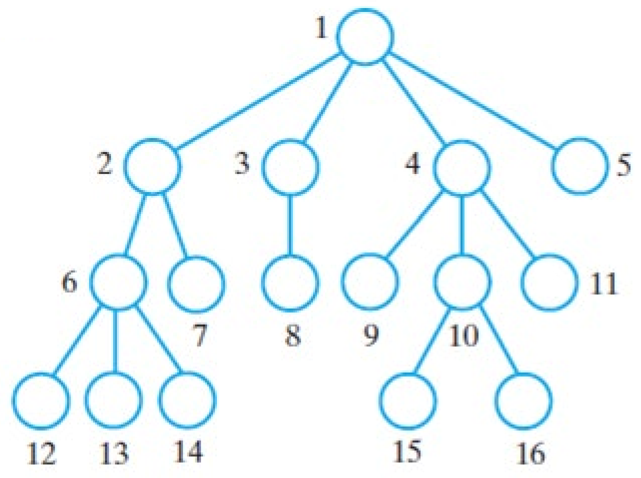


	Adjacency list	Adjacency matrix
Store graph	$O(\|V\|+\|E\|)$	$O(\|V\|^2)$
Add vertex	$O(1)$	$O(\|V\|^2)$
Add edge	$O(1)$	$O(1)$
Remove vertex	$O(\|E\|)$	$O(\|V\|^2)$
Remove edge	$O(\|E\|)$	$O(1)$
Query: are vertices *x* and *y* adjacent? (assuming that their storage positions are known)	$O(\|V\|)$	$O(1)$
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

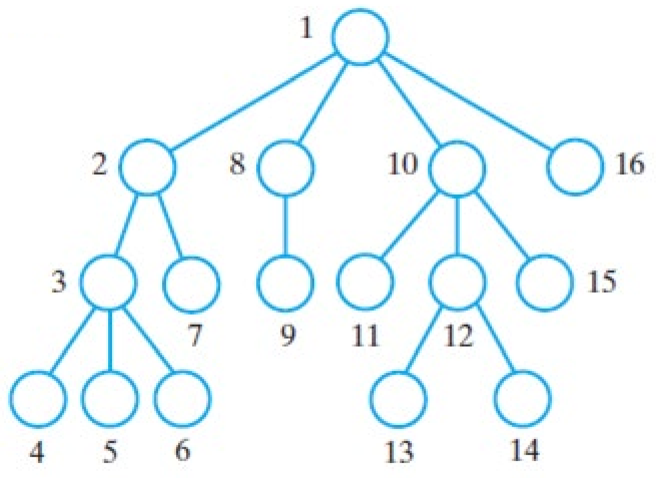
Graph Traversals

- Breadth-first search
- Depth-first search

Tree Traversals



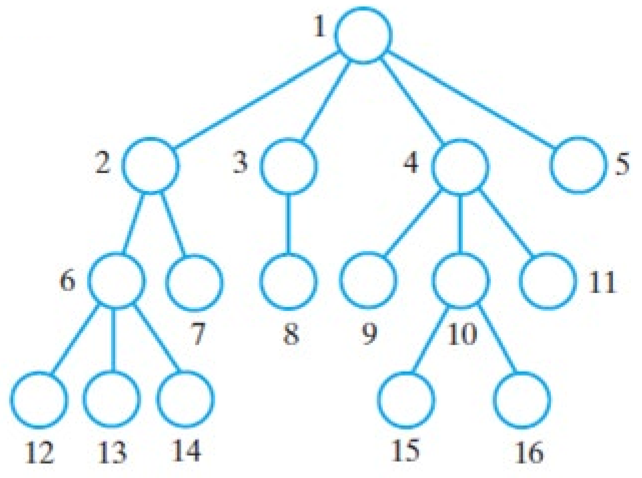
Level-order



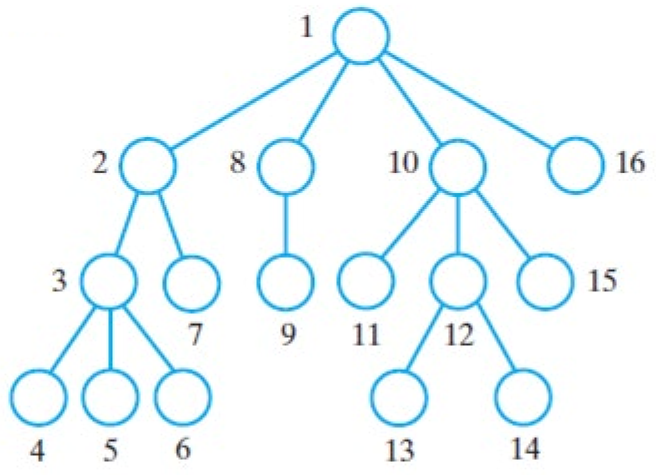
Pre-order

Graph Traversals

- When we see it as a graph



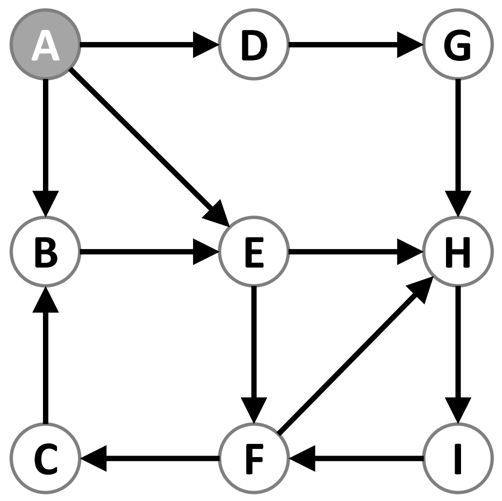
Breadth-first traversal



Depth-first traversal

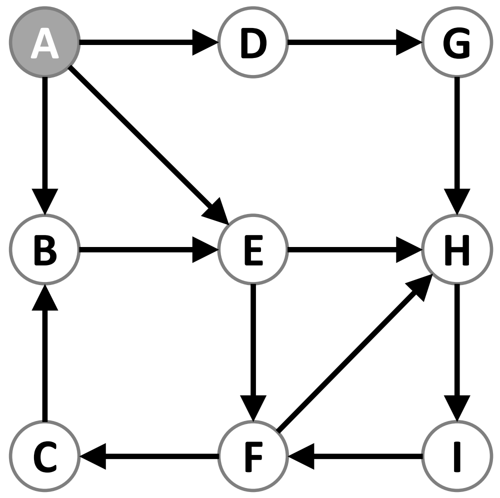
Graph Traversals

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



Breadth-First Traversal

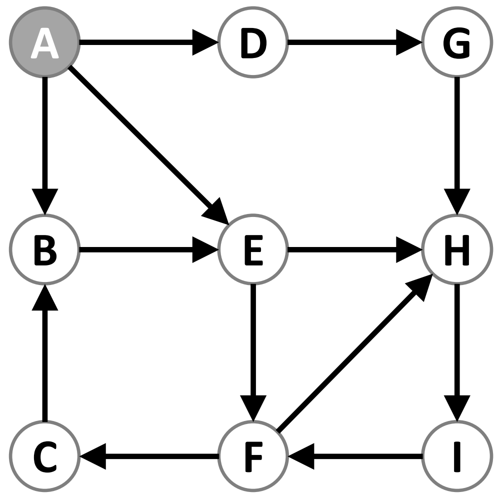
- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - Then, considers each of these neighbors and visits their neighbors.



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

Breadth-First Traversal

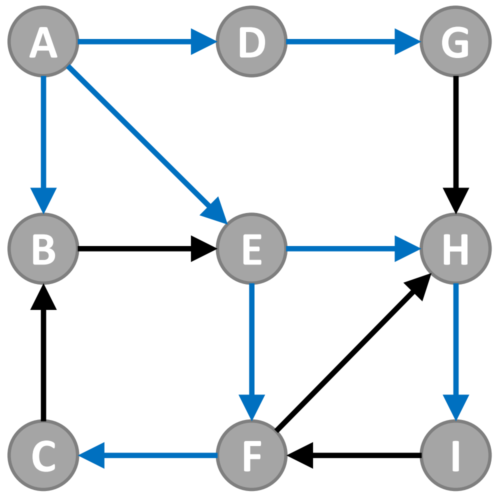
- Given an origin vertex, a breadth-first traversal
  - Visits the origin and the origin's neighbors.
  - Then, considers each of these neighbors and visits their neighbors.



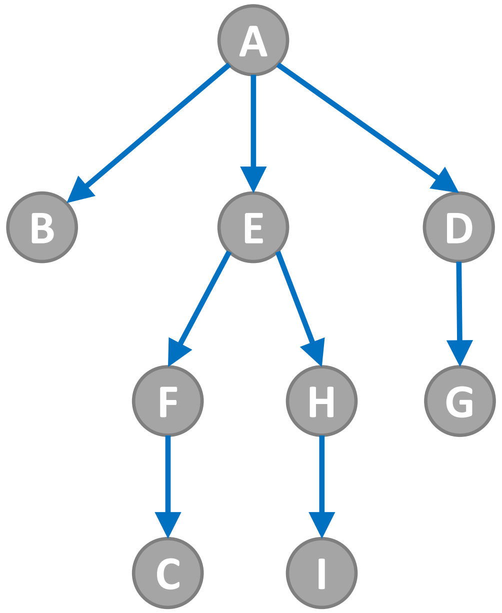
- The traversal uses a queue to hold the visited vertices.
- When we remove a vertex from this queue, we visit and enqueue the vertex's unvisited neighbors.
- Often, these queues are referred to as the **frontier** and the **explored** queues

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

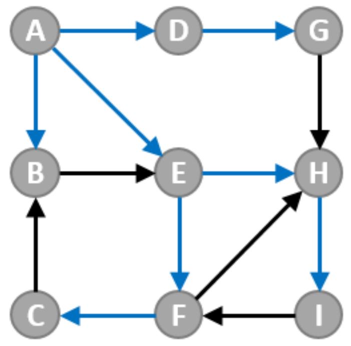


Paths form a breadth-first tree
(Order in which the nodes are expanded)

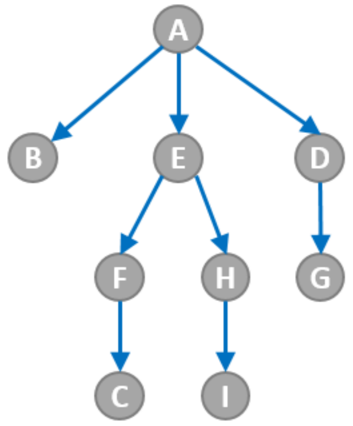
Breadth-First Traversal

- 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.
  - Either u or v will be the origin



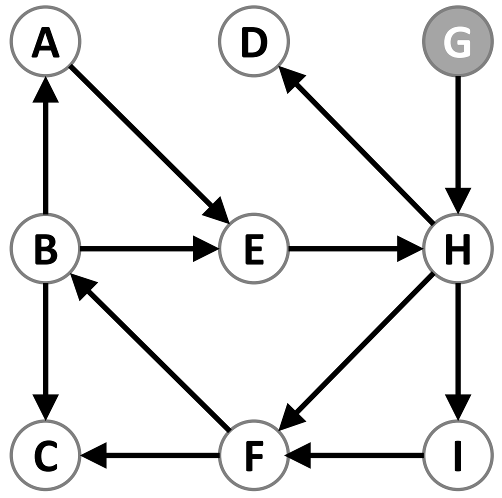
Breadth-First Traversal



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

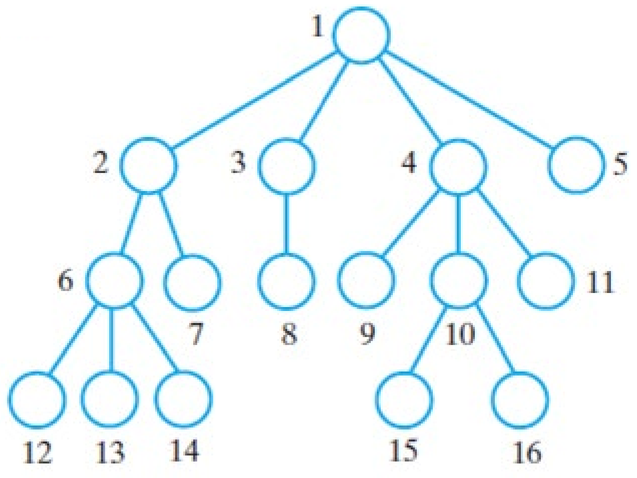
In-Class Exercise

- Perform breadth-first traversal beginning with Vertex G

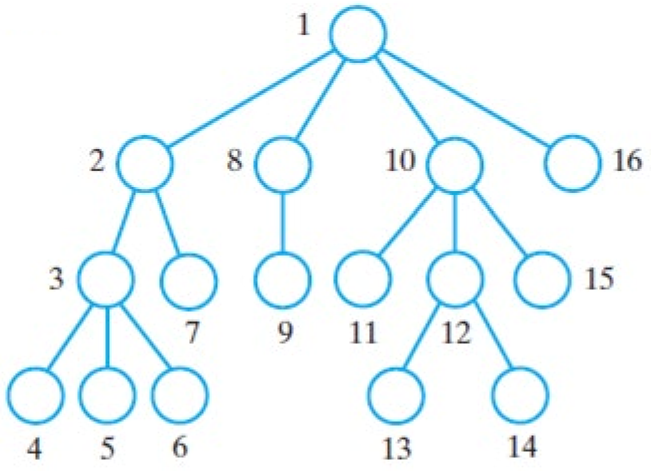


Graph Traversals

- When we see it as a graph



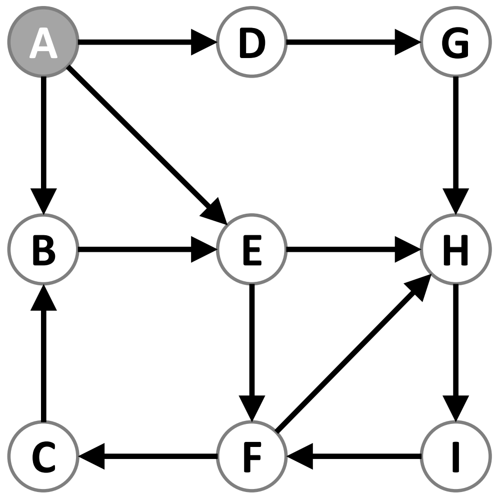
Breadth-first traversal



Depth-first traversal

Depth-First Traversal

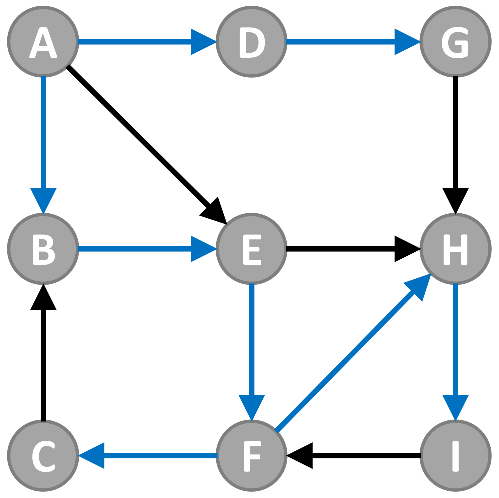
- Given an origin vertex, a depth-first traversal
  - visits the origin, then a neighbor of the origin, and a neighbor of the neighbor. It continues in this fashion until it finds no unvisited neighbor.
  - Backing up by one vertex, it considers another neighbor.

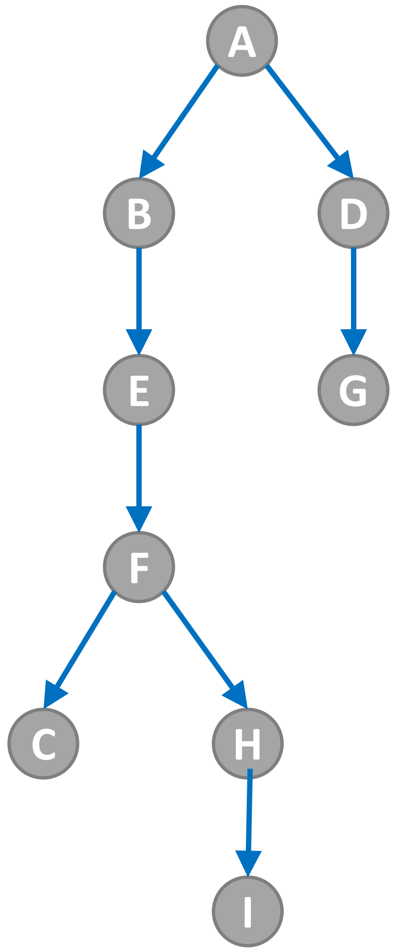


- The traversal uses a stack (can also implement it recursively) to expand the deepest unvisited nodes.

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.
  - Backs up by one vertex, it considers another neighbor.

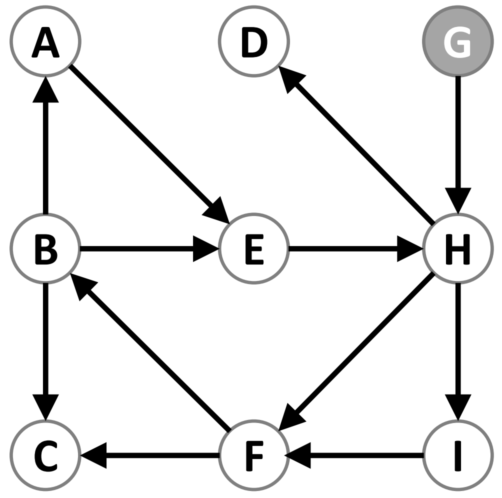




Paths form a depth-first tree
(Order in which the nodes are expanded)

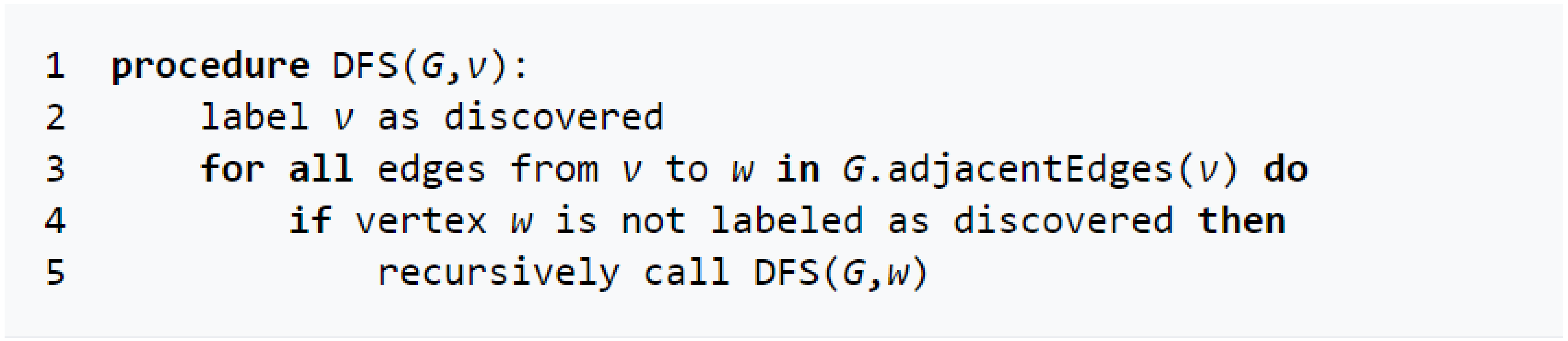
In-Class Exercise

- Perform depth-first traversal beginning with Vertex G



Implementation of DFS

- A recursive implementation of DFS

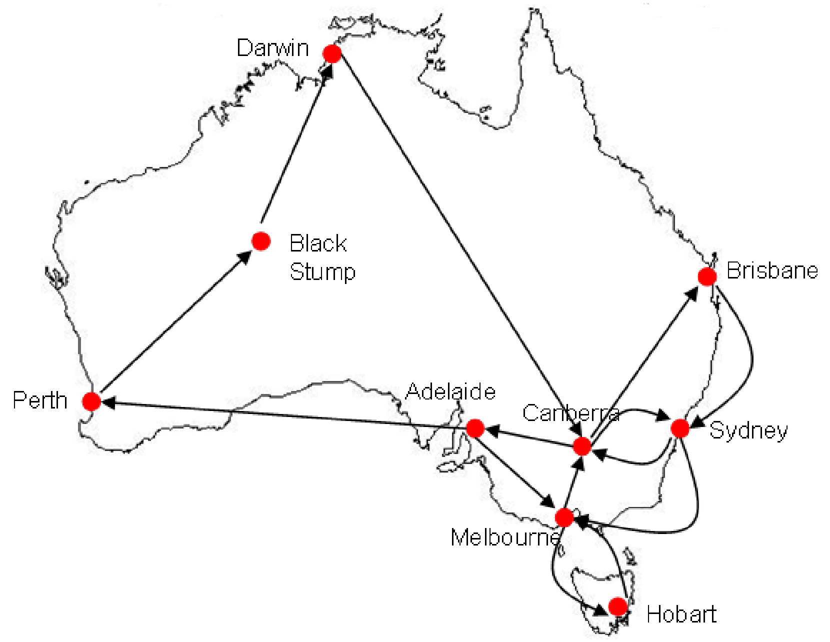


```
procedure DFS(G,v):
    label v as discovered
    for all edges from v to w in G.adjacentEdges(v) do
        if vertex w is not labeled as discovered then
            recursively call DFS(G,w)
```

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.


Darwin
Black Stump
Brisbane
Perth
Adelaide
Canberra
Sydney
Melbourne
Hobart