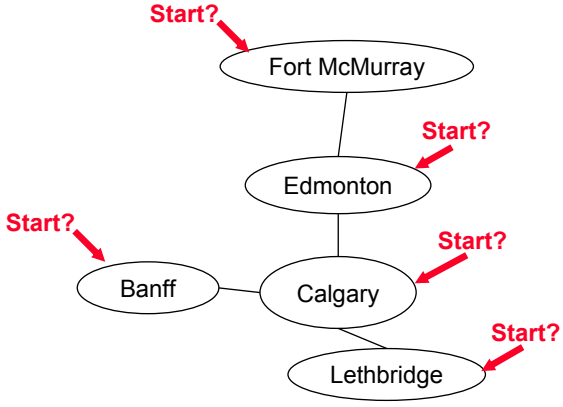
Introduction To Graphs

- In this section of notes you will learn about a new ADT: graphs.

Graphs Are Related To Trees

- Like a tree a graph consists of nodes (vertex) and arcs (edges) that connect the nodes
- Unlike a tree there is no "up/down" direction (no parent-child relation), there is no root node

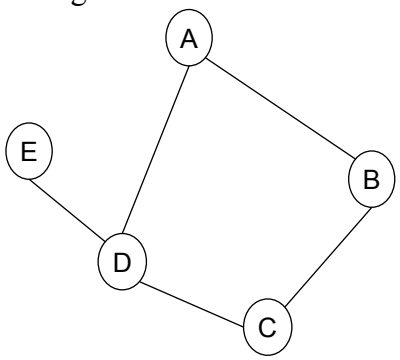


Graph Terminology

- Adjacent nodes
- Cycle
- Acyclic graph
- Sub-graph
- Connected/disconnected graphs
- Complete graphs
- Directed/undirected graphs
- Weighted graphs

Adjacent Nodes

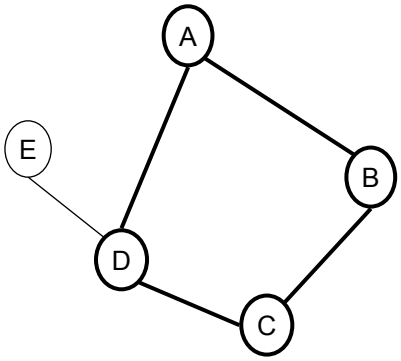
- Nodes are adjacent if they are connected by an edge



Adjacent pairs
(a, b)
(a, d)
(b, c)
(c, d)
(d, e)

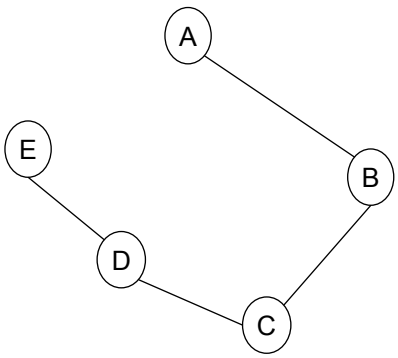
Cycle

•A path that begins and ends with the same node



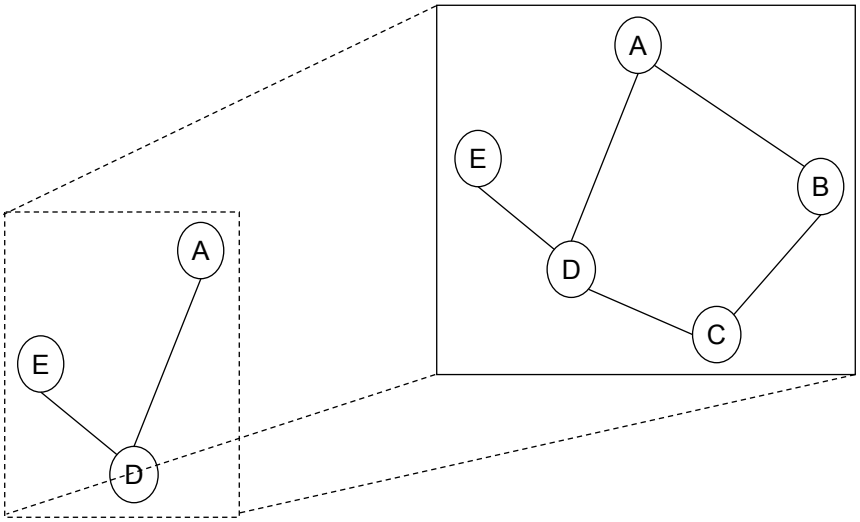
Acyclic Graph

•Has no cycles



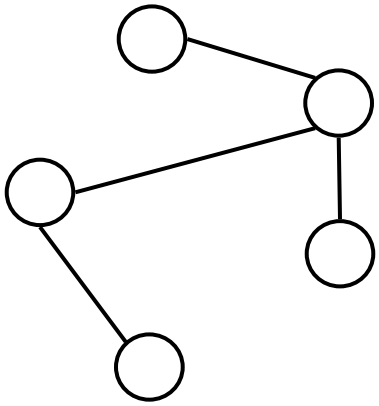
Sub-Graph

•A portion of a graph that is also a graph



Connected Graphs

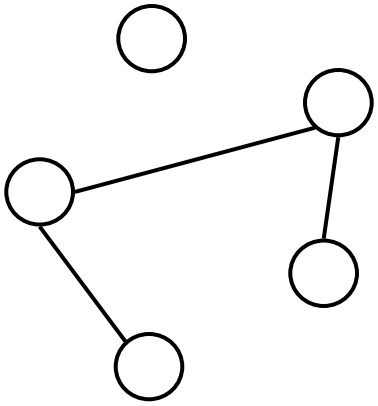
•You can go from any node to any other node by following the edges



Note: It is not a requirement for connected graphs to have edges from every pair of nodes

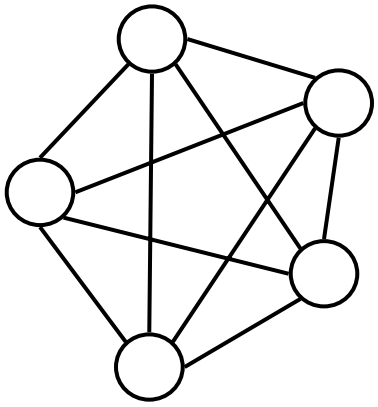
Disconnected Graphs

•Some nodes are unreachable because there is no edge that connects them with the rest of the graph



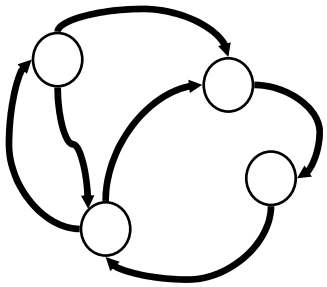
Complete Graphs

•Every pair of nodes has an edge between them (every node is directly connected to every other node)



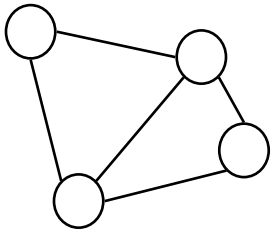
Directed Graphs

- Traversal between nodes is not guaranteed to be symmetric
  - E.g., map information that represents one way streets



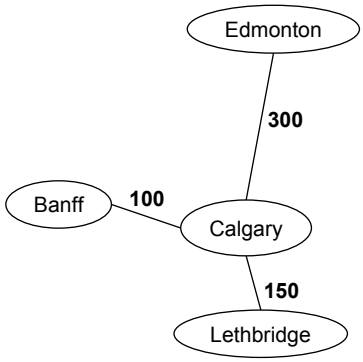
Undirected Graphs

- Each connection is symmetric (a connection in one direction guarantees a connection in other direction)



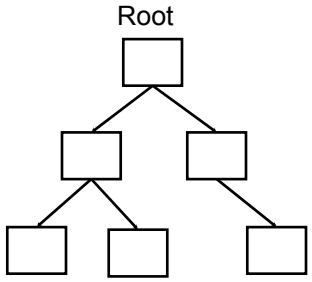
Weighted Graph

- Shows the cost of traversing an edge
- Costs of traveling between cities
  - Distance in kilometers
  - Travel time in hours
  - Dollar cost of a taxi
  - Etc.



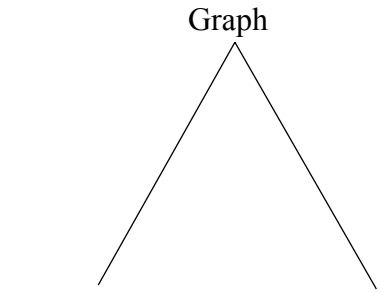
Comparing Trees And Graphs Again

- A Tree is A More Specific Form Of Graph
- A typical[1] tree is a graph that has the following characteristics
  1. It is connected
  2. It has no cycles[1]
  3. There is an up/down direction (there is a parent-child relation between nodes)
  4. One node is treated as the top (the root node has no parent node)



1 The type of tree that you were required to implement was somewhat rare

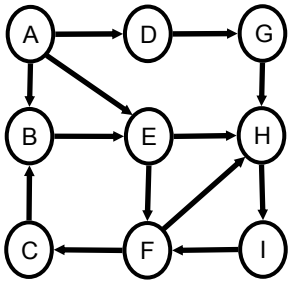
Graph Implementations



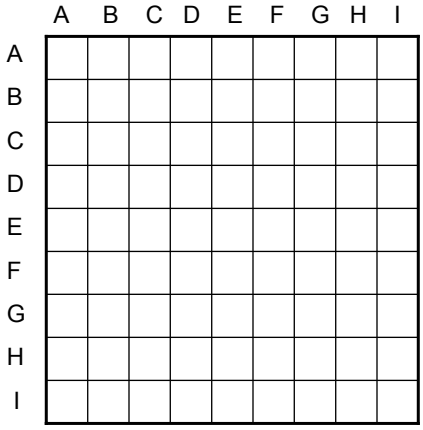
ADT (general concept)

Data structure (specific)

Adjacency Matrix: Array Implementation



ADT: Graph



Data structure: A 2D square array

- No rows = no columns = no. of nodes in the graph

Possible Array Implementations

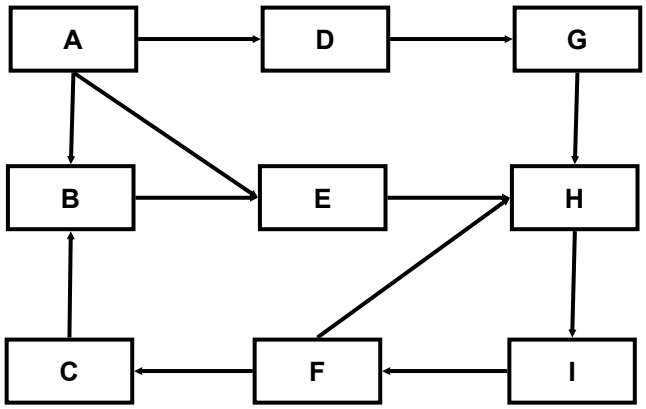
	A	B	C	D	E	F	G	H	I
A		T		T	T				
B					T				
C		T							
D							T		
E						T		T	
F			T					T	
G								T	
H									T
I						T			

A 2D array of boolean values

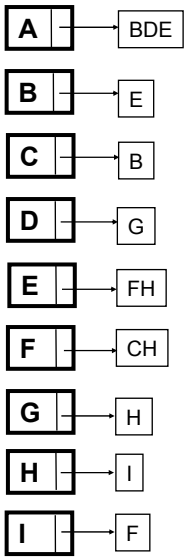
	A	B	C	D	E	F	G	H	I
A		1		1	1				
B					1				
C		1							
D							1		
E						1		1	
F			1					1	
G								1	
H									1
I						1			

A 2D array of integer values

A Linked List Implementation Of A Graph



The List Of Edges Must Be Dynamic[1]



1 Some sort of resizable list is needed e.g., a linked list or an array that can change in size

An Outline For A Node

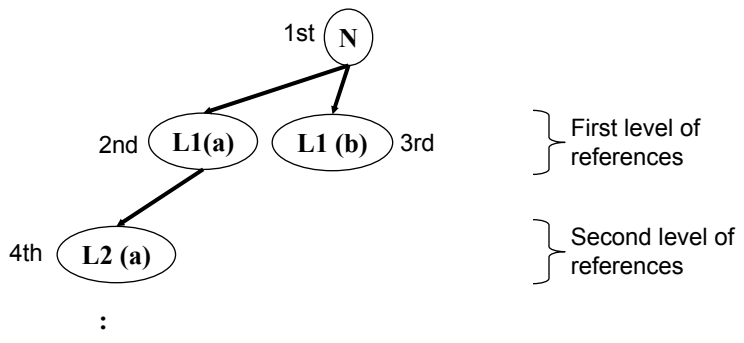
```
class Node
{
   private dataType data;
   private boolean visited;
   Dynamic list of connections;
      :        :          :
}
```

Graph Traversals

- Breadth first
- Depth first

Breadth-First Traversals

- Visit a node (N)
- Visit all of the nodes that node N refers to before following the second level of references



Algorithm For Breadth-First Traversals

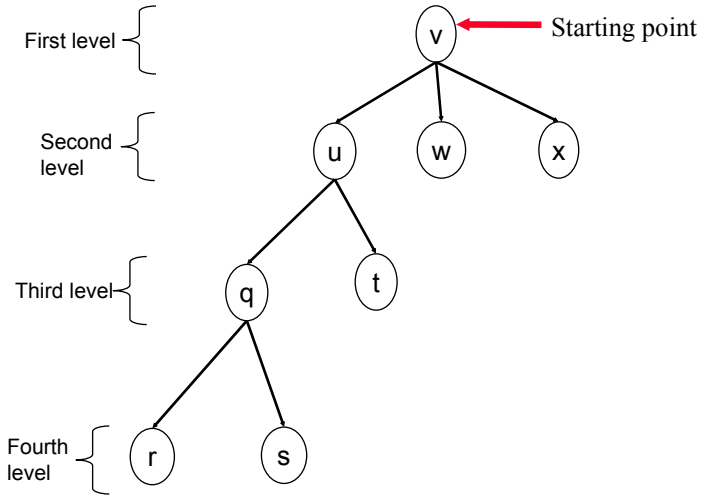
- In a fashion that is similar to breadth first traversals for trees, a queue is employed to store all the nodes that are adjacent to the node that is currently being visited.

```
breadthFirst (node)
{
    Queue nodeList = new Queue ()
    Node temp
    Mark node as visited and display node
    nodeList.enqueue(node)
```

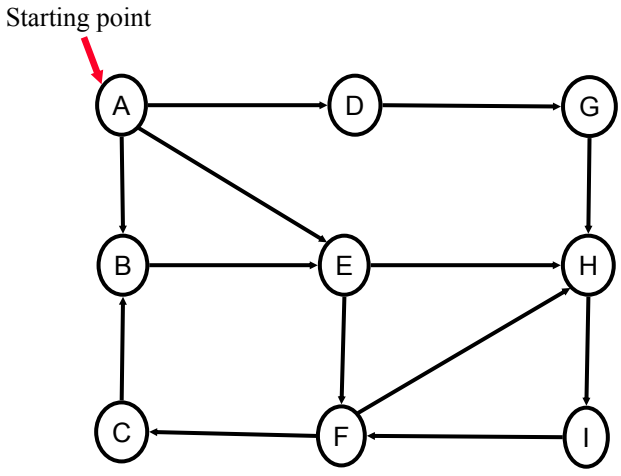

Algorithm For Breadth-First Traversals (2)

```
    while (queue.isEmpty() == false)
    {
        temp = nodeList.dequeue ()
        for (each unvisisted node uNode that is adjacent to temp)
        {
            Mark uNode as visited
            display uNode
            nodeList.enqueue(uNode)
        }
    }
}
```

First Example Of A Breadth First Traversal



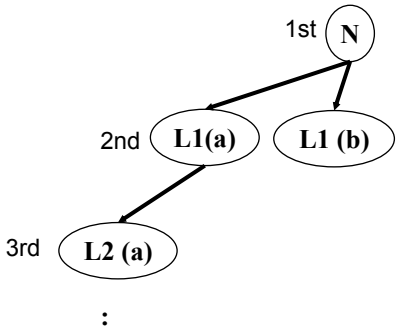
Second Example Of A Breadth-First Traversal



Q: What order do you get for a breadth-first traversal if the starting point is node E?

Depth-First Traversals

- Visit a node
- Completely follow the series of references for a chain of nodes before visiting the second reference for that node

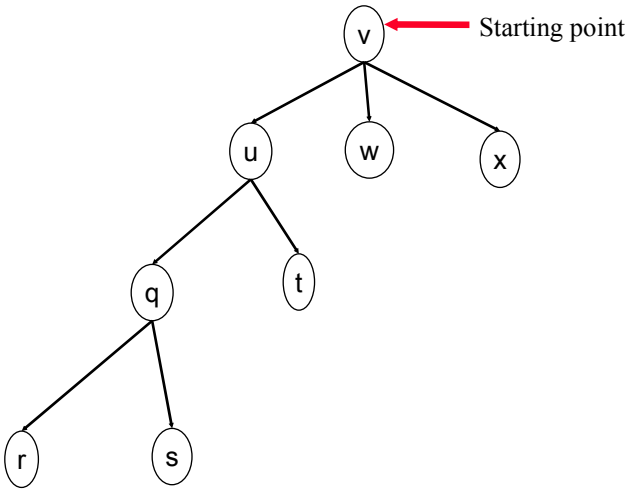


Algorithm For Depth-First Traversals

- Typically recursion is used (requires backtracking and the use of the system stack).
- If a loop is used then the programmer must create and manage his or her own stack.

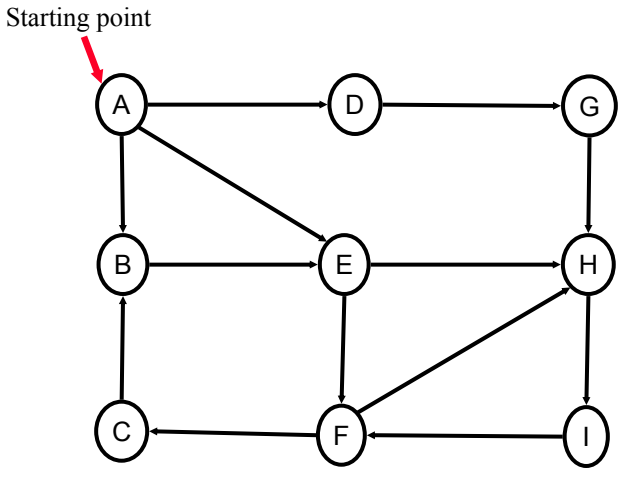
```
depthFirst (node)
{
    Display node
    Mark node as visited
    for (each unvisited node (uNode) that is adjacent to node)
        depthFirst (node)
}
```

First Example Of A Depth First Traversal



James Tam

Second Example Of A Depth-First Traversal



Q: What order do you get for a depth-first traversal if the starting point is node E?

James Tam

You Should Now Know

- What is a graph
- Common graph definitions
- What are the different ways in which graphs can be implemented
- How do breadth-first and depth-first traversals work

Sources Of Lecture Material

- *"Data Structures and Abstractions with Java"* by Frank M. Carrano and Walter Savitch
- "*Data Abstraction and Problem Solving with Java: Walls and Mirrors*" by Frank M. Carrano and Janet J. Prichard
- CPSC 331 course notes by Marina L. Gavrilova http://pages.cpsc.ucalgary.ca/~marina/331/