Trees

• Tree Structure
• Binary Tree
• Tree Traversals
The Tree Structure

- Consists of nodes and edges that organize data in a hierarchical fashion.
  - **nodes** – store the data elements.
  - **edges** – connect the nodes.
- The organization of the nodes form relationships between the data elements.
Tree Example #1
Tree Example #2
Root Node

- Topmost node of the tree.
  - Provides the single access point into the tree.
  - Has no incoming edges.
Tree Path

- The nodes encountered when following the edges from the root node to the destination node.
- Access to all other nodes must start with the root.

Nodes T, C, R, and K form a path from T to K.
Parent Node

- The node from which an incoming edge originates.
  - Every node, except the root, has a parent node.
  - A node can only have one parent.

X is the parent of nodes B and G.
Child Node

- The nodes to which outgoing edges are connected.
  - Each node can have one or more child nodes.
  - Results in a parent-child relationship.
- sibling nodes – all nodes that have the same parent.
Types of Nodes

- Nodes can be classified as either:
  - **interior node** – a node that has at least one child.
  - **leaf node** – a node that has no children.
Subtree

- A tree is by definition a recursive structure.
  - Every node can be the root of its own subtree.
  - A **subtree** consists of a subset of nodes and edges of the larger tree.
Relatives

- **descendants**
  - All nodes of a subtree are the descendants of the subtree's root.
  - Every node in the tree is a descendant of the root.

- **ancestors**
  - The ancestors of a node include all of the nodes along the node's path, excluding the node itself.
  - The root is the ancestor of all the other nodes.
The Binary Tree

- A tree in which each node can have at most two children. The nodes are commonly labeled:
  - left child
  - right child
Binary Tree Properties

- There are several properties associated with binary trees that depend on the node organization.
  - depth – the distance of a node from the root.
  - level – all nodes at a given depth share a level.
  - height – number of levels in the tree.
  - width – number of nodes on the level containing the most nodes.
  - size – number of nodes in the tree.
Binary Tree Properties
Binary Tree Properties

- Given a tree of size $n$:
  - max height = $n$
  - min height = $\lceil \log_2 n \rceil + 1$
Binary Tree Structure

- Height of a tree will be important in analyzing the efficiency of binary tree algorithms.
- Structural properties can play a role in the efficiency of an algorithm.
Full Binary Tree

- A binary tree in which each interior node contains two children.
Perfect Binary Tree

- A full binary tree in which all leaf nodes are at the same level.
Complete Binary Tree

- A binary tree of height \( h \), is a perfect binary tree down to height \( h - 1 \) and the nodes at the lowest level are filled from left to right (no gaps).
Binary Tree Implementation

- Commonly implemented as a dynamic structure in the same fashion as linked lists.
  - Can be used in many different applications.
  - Operations depend on the application.

```python
# The storage class for creating binary tree nodes.
class _BinTreeNode:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
```
Physical Implementation
Tree Traversals

- Iterates through the nodes of a tree, one node at a time in order to visit every node.
  - With a linear structure this was simple.
  - How is this done with a hierarchical structure?
    - Must begin at the root node.
    - Every node must be visited.
    - Results in a recursive solution.
Preorder Traversal

- After visiting the root,
  - traverse the nodes in the left subtree
  - then traverse the nodes in the right subtree.
Preorder Traversal

1. Visit the node.
2. Traverse the left subtree.
3. Traverse the right subtree.
Preorder Traversal

- The implementation is rather simple.
- Given a binary tree of size n, a complete traversal requires $O(n)$ to visit every node.

```python
def preorderTrav( subtree ):
    if subtree is not None :
        print( subtree.data )
        preorderTrav( subtree.left )
        preorderTrav( subtree.right )
```
Inorder Traversal

- Similar to the preorder traversal, but we traverse the left subtree before visiting the node.

1. Traverse the left subtree.
2. Visit the node.
3. Traverse the right subtree.
Inorder Traversal

- The implementation swaps the order of the visit operation and the recursive calls.

```python
def inorderTrav( subtree ):
    if subtree is not None:
        inorderTrav( subtree.left )
        print( subtree.data )
        inorderTrav( subtree.right )
```
Postorder Traversal

- Is the opposite of the preorder traversal.
  - Traverse both the left and right subtrees before visiting the node.
Breadth-First Traversal

- The nodes are visited by level, from left to right.
  - The previous traversals are all depth-first traversals.
Breadth-First Traversal

- Recursion can not be used with this traversal.
- We can use a queue and an iterative loop.

```python
def breadthFirstTrav( bintree ):
    q = Queue()
    q.enqueue( bintree )

    while not q.isEmpty() :
        # Remove the next node from the queue and visit it.
        node = q.dequeue()
        print( node.data )

        # Add the two children to the queue.
        if node.left is not None :
            q.enqueue( node.left )
        if node.right is not None :
            q.enqueue( node.right )
```