Trees

a non-introduction to horticulture

Reading LDC Ch 16
A **tree** is a non-linear, connected, *hierarchical* structure

- What would you call a collection of trees?

- It is comprised of a set of **nodes** in which elements are stored and **edges** connecting nodes

- A node can have only one **parent**, but may have multiple children
- The (only) **root** has no parent
- A **leaf** node has no children
- An **internal** node is not the root and not a leaf
- Nodes that have the same parent are **siblings**
A **subtree** is a tree structure that makes up part of another tree, rooted at some internal node.

A **path** is a sequence of edges of adjacent nodes of a tree.

- How many paths between any pair of nodes are there?

The **path length** is determined by counting the number of edges that must be followed to get from a node to another node.

A node $y$ is an **ancestor** of another node $x$ if it is on the path from the root to $x$.

Nodes that can be reached by following a path from a node $y$ to a leaf are the **descendants** of node $y$.

The **level** of a node is the length of the path from the root to the node.

The **height** of a tree is the length of the longest path from the root to a leaf.
We classify trees by the **maximum number of children** any node in the tree may have.

*General trees* have no limit to the number of children a node may have.

*n-ary trees* limit each node to no more than *n* children.

*Binary trees* are those in which nodes may have at most two children.
A tree is **balanced** if all the leaves of the tree are on the same level or at least within one level of each other.

A tree is **full** if all leaves of the tree are at the same height and every non-leaf node has exactly $n$ children (for $n$-ary trees).

A tree is **complete** if it is full, or full to the next-to-last level with all leaves at the bottom level appearing on the left side of the tree.
Tree Traversals

How do you visit all the nodes of a tree in an organized way?
Tree Traversals

Traversing a tree (visiting all nodes in a sequence) is generally more interesting (and challenging) than traversing a linear structure.

A traversal dictates the order in which the elements of a tree are assessed when starting from the root.
Nodes are visited **before** any subtrees are visited.

Visit Node
Traverse (left)
Traverse (right)

Visit the root in **between** the traversals of the left and right subtrees.

Preorder
Traverse (left)
Visit Node
Traverse (right)

Inorder

Postorder
Traverse (left)
Traverse (right)
Visit Node

Visit the root node **after** the traversals of the left and right subtrees.
Level-Order Traversal

Visit the nodes on each level, left to right, top to bottom starting at the root
Enqueue the root node of the tree
While the queue is not empty{
    Dequeue node
    Visit node
    Enqueue left child of node
    Enqueue right child of node
}

Traversals

Preorder: ABDHIEJMNCFGKL
Inorder: HDBEMJNAFCKGL
Postorder: HIDMNJEBFKLGCAN
Level-order: ABCDEFGHIJKLMNOP

Diagram of a tree with nodes labeled A to M.
List the orders!

- Preorder: A B C D E F G H
- Inorder: B D G E H C F A
- Postorder: G D H E F C B A
- Level-order: A B C D E F G H

node
Implementing Trees

- We can use **linked nodes** or **arrays**

- **Link-based** implementations are more powerful and efficient, but also more complicated
  - We will discuss it next time

- **Array-based** implementations are less obvious because arrays are linear structures and trees are not.
  - So, we need to find a way to indicate where in the array are the children of a node located
Implementing Trees

Part I: Using Arrays

Each array location holds a tree node
Computed Links in an Array

- Place tree nodes in specific indices of the array
- A node’s index can be used to calculate the indices of its parent and children

D is in index location $x = 3$.
Find indices of D’s children:
- $\text{leftChild}(x) =$
- $\text{rightChild}(x) =$
How about index of D’s parent?
- $\text{parent}(x) =$
Show the Computed-links array