CMSC 341

Introduction to Trees
Tree ADT

- Tree definition
  - A tree is a set of nodes which may be empty
  - If not empty, then there is a distinguished node $r$, called root and zero or more non-empty subtrees $T_1, T_2, \ldots T_k$, each of whose roots are connected by a directed edge from $r$.

- This recursive definition leads to recursive tree algorithms and tree properties being proved by induction.

- Every node in a tree is the root of a subtree.
A Generic Tree
Tree Terminology

- *Root* of a subtree is a child of \( r \). \( r \) is the *parent*.
- All children of a given node are called *siblings*.
- A *leaf* (or external) node has no children.
- An *internal node* is a node with one or more children.
More Tree Terminology

- A path from node \( V_1 \) to node \( V_k \) is a sequence of nodes such that \( V_i \) is the parent of \( V_{i+1} \) for \( 1 \leq i \leq k \).
- The length of this path is the number of edges encountered. The length of the path is one less than the number of nodes on the path (\( k - 1 \) in this example).
- The depth of any node in a tree is the length of the path from root to the node.
- All nodes of the same depth are at the same level.
More Tree Terminology (cont.)

- The depth of a tree is the depth of its deepest leaf.
- The height of any node in a tree is the length of the longest path from the node to a leaf.
- The height of a tree is the height of its root.
- If there is a path from $V_1$ to $V_2$, then $V_1$ is an ancestor of $V_2$ and $V_2$ is a descendent of $V_1$. 
A Unix directory tree
Tree Storage

- A tree node contains:
  - Data Element
  - Links to other nodes
- Any tree can be represented with the “first-child, next-sibling” implementation.

```java
class TreeNode {
    Object element;
    TreeNode firstChild;
    TreeNode nextSibling;
}
```
Printing a Child/Sibling Tree

// depth equals the number of tabs to indent name
private void listAll( int depth ) {
    printName( depth ); // Print the name of the object
    if( isDirectory( ) )
        for each file c in this directory (for each child)
            c.listAll( depth + 1 );
}

public void listAll( )
{
    listAll( 0 );
}

What is the output when listAll( ) is used for the Unix directory tree?
K-ary Tree

- If we know the maximum number of children each node will have, K, we can use an array of children references in each node.

```java
class KTreeNode {
    Object element;
    KTreeNode children[ K ];
}
```
Pseudocode for Printing a K-ary Tree

// depth equals the number of tabs to indent name
private void listAll( int depth )
{
    printElement( depth ); // Print the value of the object
    if( children != null )
        for each child c in children array
            c.listAll( depth + 1 );
}

public void listAll( )
{
    listAll( 0 );
}
Binary Trees

- A special case of K-ary tree is a tree whose nodes have exactly two children pointers -- binary trees.

- A *binary tree* is a rooted tree in which no node can have more than two children AND the children are distinguished as *left* and *right*. 
The Binary Node Class

private static class BinaryNode<AnyType>
{
    // Constructors
    BinaryNode( AnyType theElement )
    {
        this( theElement, null, null );
    }

    BinaryNode( AnyType theElement, BinaryNode<AnyType> lt,
                BinaryNode<AnyType> rt )
    {
        element  = theElement; left = lt; right = rt;
    }

    AnyType element;            // The data in the node
    BinaryNode<AnyType> left;   // Left child
    BinaryNode<AnyType> right;  // Right child
}
A full Binary Tree is a Binary Tree in which every node either has two children or is a leaf (every interior node has two children).
FBT Theorem

- Theorem: A FBT with \( n \) internal nodes has \( n + 1 \) leaf nodes.
- Proof by strong induction on the number of internal nodes, \( n \):
  - Base case:
    - Binary Tree of one node (the root) has:
      - zero internal nodes
      - one external node (the root)
  - Inductive Assumption:
    - Assume all FBTs with up to and including \( n \) internal nodes have \( n + 1 \) external nodes.
FBT Proof (cont’d)

- Inductive Step - prove true for a tree with \( n + 1 \) internal nodes (i.e. a tree with \( n + 1 \) internal nodes has \( (n + 1) + 1 = n + 2 \) leaves)
  - Let \( T \) be a FBT of \( n \) internal nodes.
  - It therefore has \( n + 1 \) external nodes. (Inductive Assumption)
  - Enlarge \( T \) so it has \( n+1 \) internal nodes by adding two nodes to some leaf. These new nodes are therefore leaf nodes.
  - Number of leaf nodes increases by 2, but the former leaf becomes internal.
  - So,
    - \# internal nodes becomes \( n + 1 \),
    - \# leaves becomes \( (n + 1) + 1 = n + 2 \)
Perfect Binary Tree

- A *Perfect Binary Tree* is a full Binary Tree in which all leaves have the same depth.
PBT Theorem

- **Theorem:** The number of nodes in a PBT is $2^{h+1}-1$, where $h$ is height.

- Proof by strong induction on $h$, the height of the PBT:
  - Notice that the number of nodes at each level is $2^l$.
    (Proof of this is a simple induction - left to student as exercise). Recall that the height of the root is 0.
  - **Base Case:**
    The tree has one node; then $h = 0$ and $n = 1$ and $2^{(h+1)} = 2^{(0+1)} - 1 = 2^1 - 1 = 2 - 1 = 1 = n$.
  - **Inductive Assumption:** Assume true for all PBTs with height $h \leq H$. 
Proof of PBT Theorem (cont)

- Prove true for PBT with height H+1:
  - Consider a PBT with height H + 1. It consists of a root and two subtrees of height H. Therefore, since the theorem is true for the subtrees (by the inductive assumption since they have height = H)
    - $(2^{(H+1)} - 1)$ for the left subtree
    - $(2^{(H+1)} - 1)$ for the right subtree
    - 1 for the root
  - Thus, $n = 2 \times (2^{(H+1)} - 1) + 1$
    - $= 2^{((H+1)+1)} - 2 + 1 = 2^{((H+1)+1)} - 1$
Complete Binary Trees

- Complete Binary Tree
- A *complete Binary Tree* is a perfect Binary Tree except that the lowest level may not be full. If not, it is filled from left to right.
Tree Traversals

- Inorder
- Preorder
- Postorder
- Levelorder
Constructing Trees

Is it possible to reconstruct a Binary Tree from just one of its pre-order, inorder, or post-order sequences?
Constructing Trees (cont)

- Given two sequences (say pre-order and inorder) is the tree unique?
How do we find something in a Binary Tree?

- We must recursively search the entire tree. Return a reference to node containing x, return NULL if x is not found

```java
BinaryNode<AnyType> find( Object x) {
    BinaryNode<AnyType> t = null; // found it here
    if ( element.equals(x) ) return element;

    // not here, look in the left subtree
    if(left != null)
        t = left.find(x);

    // if not in the left subtree, look in the right subtree
    if ( t == null)
        t = right.find(x);

    // return pointer, NULL if not found
    return t;
}
```
Binary Trees and Recursion

- A Binary Tree can have many properties
  - Number of leaves
  - Number of interior nodes
  - Is it a full binary tree?
  - Is it a perfect binary tree?
  - Height of the tree

- Each of these properties can be determined using a recursive function.
Recursive Binary Tree Function

return-type function (BinaryNode<AnyType> t) {
    // base case – usually empty tree
    if (t == null) return xxxx;

    // determine if the node pointed to by t has the property

    // traverse down the tree by recursively “asking” left/right children
    // if their subtree has the property

    return theResult;
}
boolean isFBT (BinaryNode<AnyType> t) {
    // base case – an empty tree is a FBT
    if (t == null) return true;

    // determine if this node is “full”
    // if just one child, return – the tree is not full
    if ((t.left && !t.right) || (t.right && !t.left))
        return false;

    // if this node is full, “ask” its subtrees if they are full
    // if both are FBTs, then the entire tree is an FBT
    // if either of the subtrees is not FBT, then the tree is not
    return isFBT( t.right ) && isFBT( t.left );
}
Other Recursive Binary Tree Functions

- Count number of interior nodes
  ```
  int countInteriorNodes(BinaryNode<AnyType> t);
  ```

- Determine the height of a binary tree. By convention (and for ease of coding) the height of an empty tree is -1
  ```
  int height(BinaryNode<AnyType> t);
  ```

- Many others
Other Binary Tree Operations

- How do we insert a new element into a binary tree?
- How do we remove an element from a binary tree?