CS206

Trees
Copying

ArrayList x = new ArrayList();
ArrayList y = x;

see project L1029 class CopyObject
Tree

- A tree is an abstract model of a hierarchical structure
- Nodes have a parent-child relation
Terminology

- **root**: no parent – A
- **external node/leaf**: no children – E, I, J, K, G, H, D
- **internal node**: - node with at least one child - A, B, C, F
- **ancestor/descendent**
- **depth**: # of ancestors
- **Height**: max depth

**Subtree**: tree consisting of a node and its descendants
Binary Tree

- An ordered tree with every node having at most two children – left and right
Type of Binary Trees

- A binary tree is proper (or full) if each node has zero or two children.
- A binary tree is complete if every level (except possibly the last) is filled.
- If a complete binary tree is filled at every level, it is perfect.

![Binary Tree Diagrams]
Binary Tree Properties

- Let $n$ denote the number of nodes and $h$ the height of a binary tree
  - $h + 1 \leq n \leq 2^{h+1} - 1$
  - $\log(n + 1) - 1 \leq h \leq n - 1$

- Height of a binary tree is usually $O(\log n)$ of the max number of nodes — true worst case $O(1)$
public interface BinaryTreeInterface<E extends Comparable<E>>
{
    int size();
    int maxDepth();
    boolean isEmpty();
    boolean contains(E element);
    void insert(E element);
    E remove(E element);
}
Implementation

private class Node<E> {
    public E element;
    public Node<E> left;
    public Node<E> right;
    // constructors
    public boolean isLeaf() {
        // ?
    }
}

---

CS206 9  Lec13
public class LinkedBinaryTree<E extends Comparable<E>> implements BinaryTree<E> {

    // what instance variables?
    // nested Node class

}
Insertion

- smaller to the left, bigger to the right
Draw some Binary Trees

- 11, 6, 8, 19, 4, 10, 5, 17, 43, 49, 31
- 6, 19, 10, 5, 43, 31, 11, 8, 4, 17, 49
- 4, 5, 6, 49, 43, 31, 19, 10, 11, 8, 17
- 17, 31, 8, 19, 43, 11, 5, 49, 10, 6, 4
contains

- boolean contains(E element);
- returns true if found in the tree, false otherwise
Algorithm

- compare with root of **current subtree**
  - root is empty – return false
  - root == element – return true
  - root < element – recurse on right child
  - root > element - recurse on left child
findRec(root, key):
    if root == null:
        return false
    if root.key == key:
        return true
    if root.key > key:
        return findRec(root.left, key)
    else
        return findRec(root.right, key)
Recursive Helper Method

- The signature of `contains` doesn’t allow any `Node` references (it cannot since `Node` is private)

- so define a private, recursive “helper” method.

```java
public boolean contains(E element) {
    if (root==null) return false;
    return iContains(root, element)!==null;
}

private Node iContains(Node treepart, E toBeFound) {
    ...
}
```
private Node iContains(Node treepart, E toBeFound)
{
    if (treepart==null) return null;
    int cmp = treepart.element.compareTo(toBeFound);
    if (cmp==0)
    {
        return treepart;
    }
    else if (cmp<0)
    {
        return iContains(treepart.left, toBeFound);
    }
    else // cmp>0
    {
        return iContains(treepart.right, toBeFound);
    }
}
insert

- void insert(E element);
- new node is always inserted as a leaf
- inserts to
  - left subtree if element is smaller than subtree root
  - right subtree if larger
Pseudo Code

insertRec(node, key):
    if node == null:
        add key to tree
    if root.key > key:
        node.left =
        insertRec(node.left, key)
    else
        node.right =
        insertRec(node.right, key)
Insert, with a helper

```java
public void insert(E element) {
    size++;
    if (root == null) {
        root = new Node(element);
        return;
    }
    iInsert(root, element);
}

private void iInsert(Node treepart, E toBeAdded) {
    ...
}
```
```java
private void iInsert(Node treepart, E toBeAdded) {
    int cmp = treepart.element.compareTo(toBeAdded);
    if (cmp==0) {
        return; // the item is in the tree
    } else if (cmp<0) {
        if (treepart.left==null) {
            treepart.left=new Node(toBeAdded);
        } else {
            iInsert(treepart.left, toBeAdded);
        }
    } else { // cmp>0
        if (treepart.right==null) {
            treepart.right=new Node(toBeAdded);
        } else {
            iInsert(treepart.right, toBeAdded);
        }
    }
}
```
Height / maxDepth

Again, using a recursive helper method

```java
@Override
public int maxDepth()
{
    return iMaxDepth(root, 1);
}

int iMaxDepth(Node n, int depth) {
    ...
}