CS206

Trees

Part 2

March 24
Binary Tree

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

```
        -
       / \
      +   +
     /    /
    *    *
   /     /
  +     +
 /     /  \
*     6   -
 /     /   /
3     3   7
 /   /   /   \
1   3   4   -
```

public interface BinaryTreeInterface<B> {
    int size();
    boolean isEmpty();
    boolean contains(B element);
    void insert(B element);
    int height();
    B remove(B element);
}
private class Node {
    E payload;
    Node right;
    Node left;

    public Node(E e) {
        payload = e;
        right = null;
        left = null;
    }

    public String toString() {
        return payload.toString();
    }
}
public class LinkedBinaryTree<E extends Comparable<E>> implements BinaryTreeInterface<E>
{
    /** The number of elements in the tree (optional but useful) */
    private int size;

    /** The root of the tree */
    private Node root;

size() and isEmpty()

```java
@Override
public int size() {
    return size;
}

@Override
public boolean isEmpty() {
    return size==0;
}
```
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
Pseudo Code

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) {
    ...
}
```
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 node.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) {
    ...
}
```
size (again)

• Suppose the LinkedBinary tree class did not keep size instance variable.

• Need new implementations of isEmpty() and size()

```java
@Override
public boolean isEmptyAlt() {
    return root == null;
}
```
public int sizeAlt() {
    return iSize(root);
}

private int iSize(Node treepart) {
    if (treepart == null) return 0;
    return 1 + iSize(treepart.left) + iSize(treepart.right);
}
/**
 * Recursive helper function for determining if an element is in tree.
 * @param treepart the root of the current subtree to examine
 * @param toBeFound the element being searched for
 * @return true iff the element is in the tree.
 */

private Node iContains(Node treepart, E toBeFound) {
    int cmp = treepart.payload.compareTo(toBeFound);
    if (cmp==0) return treepart;
    if (cmp<0) {
        if (treepart.left==null) return null;
        else return iContains(treepart.left, toBeFound);
    }
    else {
        if (treepart.right==null) return null;
        else {
            return iContains(treepart.right, toBeFound);
        }
    }
}
“in class” exercise

- Complete the implementation of iInsert using pencil and paper **only**
- Strive to be correct
- Think
- Take a picture of your hand written code and send it to me