CS 310: Binary Tree Traversals

Chris Kauffman

Week 10-2
Logistics

Reading

- Weiss Ch. 7 Recursion
- Weiss Ch. 19 BSTs

Last Time

- What’s a tree?
- size(Node n) : how many nodes in tree rooted at n?
- height(Node n) : height of tree rooted at n?
- Why does recursion rock?
IN RANGE IN CURRENT
ca = BM <-->| 1 5 x x | <--> | 6 8 x x | <--> EM
    *current*
ca.add(4)
ca = BM <-->| 1 4 5 x | <--> | 6 8 x x | <--> EM

CURRENT SMALLER
ca = BM <-->| 1 x x x | <--> | 5 6 7 x | <--> EM
    *current*
ca.add(4)
ca = BM <-->| 1 4 x x | <--> | 5 6 7 x | <--> EM

CURRENT.NEXT SMALLER
ca = BM <-->| 1 2 3 x | <--> | 5 6 x x | <--> EM
    *current*
ca.add(4)
ca = BM <-->| 1 2 3 x | <--> | 4 5 6 x | <--> EM

TIE GOES TO CURRENT.NEXT
ca = BM <-->| 1 2 x x | <--> | 5 6 x x | <--> EM
    *current*
ca.add(4)
ca = BM <-->| 1 2 x x | <--> | 4 5 6 x | <--> EM
Number 3 Notes: Class Structure

ArrayNode<T>
- Array of Object for data
- All T must be Comparable

ChainedArrays<T>
- Head/Tail of ArrayNode<T>
- Each ArrayNode has same capacity

MyListNode<T> (T is comparable, MyListNode is comparable)
MyHashTable<T> (Essentially a HashSet)
- Array of ChainedArrays<MyListNode<T>> for data
- Head/Tail of MyListNode<T> for data
- rehash() when load > 1 (element count > table size)

Entry<K,V> (K is comparable, Entry is comparable)
MyHashMap<K,V> (Essentially a Map from Set)
- Wrapper for MyHashTable<Entry<K,V> >

Order (buy/sell/completed trade)
StockExchange
- MyHashMap<String, ArrayList<Order> > (completed trades)
- ArrayList<ArrayList<Order> > (unmatched buy/sell orders)
The Many Ways to Walk

No list property: several orders to traverse tree

- (a) Pre-order traversal (parent, left, right)
- (b) Post-order traversal (left, right, parent)
- (c) In-order traversal (left, parent, right)

Picture shows the order nodes will be visited in each type of traversal
Walk This Tree

Show

- (a) Pre-order traversal (parent, left, right)
- (b) Post-order traversal (left, right, parent)
- (c) In-order traversal (left, parent, right)

Which one "sorts" the numbers?
Implementing Traversals for Binary Trees

class Tree<T>{
    private Node<T> root;

    public void printPreOrder(){
        preOrder(this.root);
    }
    private static void
    preOrder(Node<T> t){
        ... print(t.data) ...
    }

    public void printInOrder(){ }
    private static void
    inOrder(Node<T> t){ }

    public void printPostOrder(){ }
    private static void
    postOrder(Node<T> t){ }
}

class Node<T> { 
    T data;
    Node<T> left, right;
}

Implement Print Traversals

- preOrder(this.root)
- postOrder(this.root)
- inOrder(this.root)

2 Ways

- Recursively (first)
- Iteratively (good luck...)
Recursive Implementation of Traversals

```java
inOrder(Node t) {
    if (t != null) {
        inOrder(t.left);
        print(t.data);
        inOrder(t.right);
    }
}

preOrder(Node t) {
    if (t != null) {
        print(t.data);
        preOrder(t.left);
        preOrder(t.right);
    }
}

postOrder(Node t) {
    if (t != null) {
        postOrder(t.left);
        postOrder(t.right);
        print(t.data);
    }
}
```

Evaluate

- Correct?
- Time complexity?
- Space complexity?
- What makes this so easy?
Today’s code distribution contains demos of recursive methods

**SimpleList.java**
Demos recursive version of list length

**Tree.java**
Contains a very simple tree example that demos
- `size()`
- `height()`
- Traversals: Pre-order, In-order, Post-order

**JGrasp helpful**
- Visualize list/tree
- Step through recursive methods
- Use debugger to watch call stack and position in tree