CS 310: Recursion and Tree Traversals

Chris Kauffman

Week 10-1
Logistics

HW 3

- Up and due next week Sunday (11/15)
- Sorted Chained Array
  - Sorted fat nodes, overall sorted
  - Provides an iterator
- Build a Hash Map
  - Separately chained
  - Overlay linked list for iterator to traverse
- Simple stock trading application

Reading

- Weiss Ch 18 Trees
  - Ch 18.4: - Iterator classes for traversals
- Weiss Ch 19 Binary Search Trees
- Weiss Ch. 7 Recursion

Today

- Tree Traversals
- Recursive traversals
- Recursion practice for tree properties
Next the 2 weeks we’ll talk about roots
For simplicity, we’ll call them trees
Mutated Nodes

Node structures should be familiar for linked lists

- Singly linked: `next/data`
- Doubly linked: `next/previous/data`

Trees use Nodes as well

- children, data, possibly parent
- Arbitrary Trees: List<Node> of children
- Binary Trees: left and right children
Tree Properties of Interest

- Root of tree
- Leaves
- Data at nodes

- Size (number of nodes)
- Height of tree
- Depth of a node

<table>
<thead>
<tr>
<th>Node</th>
<th>Height</th>
<th>Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>H</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>I</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>J</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>K</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>
You spend years writing code without recursion and then one day you have to write functions that operate on trees and realize recursion is amazing.

–Kevin DeRonne
Recursion Warm-Up

Write two versions of

int length(Node)

1. Iterative
2. Recursive

Compare and contrast

class Node<T> {
    T data; Node<T> next;
    public Node(T d, Node<T> n){
        this.data=d; this.next=n;
    }
}

// Singly linked
// No header/auxiliary/dummy nodes
class SimpleList<T>{
    Node<T> head; // null When empty
    public int length(){
        return length(this.head);
    }
    public static <T>
    int length(Node<T> n){
        // Iterative version?
        // Recursive version?
    }
}
Tree Versions: Define two methods, Recursive First

### Tree Nodes

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

**Usage**

```java
Tree<Integer> myTree = new Tree();
// add some stuff to myTree
int s = myTree.size();
int h = myTree.height();
```

```java
int size(Node<T> t)
Number of nodes in tree t
```

```java
public Tree<T>{
    Node<T> root;
    public int size(){
        return size(this.root);
    }
    // Total number of nodes
    public static <T>
    int size( Node<T> t ){
        // Recursive version?
    }
}
```

```java
int height(Node<T> t)
Depth of deepest node in t
```

```java
public Tree<T>{
    Node<T> root;
    public int height(){
        return height(this.root);
    }
    // Depth of deepest node
    public static <T>
    int height( Node<T> t ){
        // Recursive version?
    }
}
```
Recursive Implementations

Slight difference of definitions from textbook

- Empty tree has size=0 and height=0
- 1-node tree has size=1 and height=1

```java
int size(Node<T> t)
Number of nodes in tree t
// Total number of nodes
public static <T>
int size(Node<T> t) {
    if (t == null) {
        return 0;
    }
    int sL = size(t.left);
    int sR = size(t.right);
    return 1 + sL + sR;
}
```

```java
int height(Node<T> t)
Depth of deepest node in t
// Depth of deepest node
public static <T>
int height(Node<T> t) {
    if (t == null) {
        return 0;
    }
    int hL = height(t.left);
    int hR = height(t.right);
    int bigger = Math.max(hL, hR);
    return 1 + bigger;
}
```
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