CompSci 201, L18: Tree Recursion
Person in CS: Kathleen Booth

- 1922 – 2022
- British Mathematician, PhD in 1950
- Worked to design the first *assembly language* for early computer designs in the 1950s
- May have been the first woman to write a book on programming
- Early interest in *neural networks*
Logistics, Coming up

• P4 Autocomplete due today 10/31

• APT8 (Tree problems) due this Wed., 11/2

• P5: Huffman releasing this week, due Monday 11/14
Three ways to recursively traverse a binary tree (search or not)

- Difference is in where the non-recursive part is

<table>
<thead>
<tr>
<th>inOrder</th>
<th>preOrder</th>
<th>postOrder</th>
</tr>
</thead>
<tbody>
<tr>
<td>void inOrder(TreeNode t) {</td>
<td>void preOrder(TreeNode t) {</td>
<td>void postOrder(TreeNode t) {</td>
</tr>
<tr>
<td>if (t != null) {</td>
<td>if (t != null) {</td>
<td>if (t != null) {</td>
</tr>
<tr>
<td>inOrder(t.left);</td>
<td>System.out.println(t.info);</td>
<td>postOrder(t.left);</td>
</tr>
<tr>
<td>System.out.println(t.info);</td>
<td>preOrder(t.left);</td>
<td>postOrder(t.right);</td>
</tr>
<tr>
<td>inOrder(t.right);</td>
<td>preOrder(t.right);</td>
<td>System.out.println(t.info);</td>
</tr>
<tr>
<td>}</td>
<td>}</td>
<td>}</td>
</tr>
</tbody>
</table>

```java
void inOrder(TreeNode t) {
    if (t != null) {
        inOrder(t.left);
        System.out.println(t.info);
        inOrder(t.right);
    }
}
```

```java
void preOrder(TreeNode t) {
    if (t != null) {
        System.out.println(t.info);
        preOrder(t.left);
        preOrder(t.right);
    }
}
```

```java
void postOrder(TreeNode t) {
    if (t != null) {
        postOrder(t.left);
        postOrder(t.right);
        System.out.println(t.info);
    }
}
```
Wrapper and recursive helper method to return List

```java
public ArrayList<String> visit(TreeNode root) {
    ArrayList<String> list = new ArrayList<>();
    doInOrder(root, list);
    return list;
}

private void doInOrder(TreeNode root, ArrayList<String> list) {
    if (root != null) {
        doInOrder(root.left, list);
        list.add(root.info);
        doInOrder(root.right, list);
    }
}
```

- In order traversal → list?
- Create list, call helper, return list
- values in returned list in order
Problem Statement

Write a method that returns the number of nodes of a binary tree. The TreeNode class will be accessible when your method is tested.

is characterized by the pre-order string 8, 4, x, 6, x, x, 12, 10, x, x, 15, x, x

```java
public class TreeCount {
    public int count(TreeNode tree) {
        // replace with working code
        return 0;
    }
}
```
Solving TreeCount in Picture & Code

6 = 1 + (2 + 3)

public int count(TreeNode tree) {
    if (tree == null) {
        return 0;
    }
    return 1 + count(tree.left) + count(tree.right);
}

3 = 1 + (1 + 1)

2 = 1 + (0 + 1)

1 = 1 + (0 + 0)

1 = 1 + (0 + 0)
Go to duke.is/8xcjw

Not graded for correctness, just participation.

Try to answer *without* looking back at slides and notes.

But do talk to your neighbors!
Complexity of tree traversal

• Intuition: visit every node once and print it
  • If there are N nodes, should be O(N)
  • But what about recursive calls?

• More generally/formally:
  • We create a recurrence relation (an equation)
  • Solving the equation yields runtime
Analyzing Recursive Runtime

Develop a recurrence relation of the form

\[ T(N) = a \cdot T(g(N)) + f(N) \]

Where:

- \( T(N) \) - runtime of method with input size \( N \)
- \( a \) is the number of recursive calls
- \( g(N) \) - how much input size decreases on each recursive call
- \( f(N) \) - runtime of non-recursive code on input size \( N \)
LinkedList Example: Runtime of Recursive Reverse

```java
public Node reverse(Node list) {
    if (list == null || list.next == null) {
        return list;
    }
    // in A->B->C->D, what does A point at?
    // afterMe -> D->C->B->null
    Node afterMe = reverse(list.next);
    list.next.next = list;
    list.next = null;
    return afterMe;
}
```

\[
T(N) = T(N - 1) + O(1)
\]

\[
g(N) = N - 1
\]

\[
f(N) = O(1)
\]
Solving Recurrence Relation

\[
T(N) = T(N - 1) + O(1) \\
= (T(N - 2) + O(1)) + O(1) \\
= (T(N - 3) + 3 \cdot O(1)) + O(1) \\
\vdots \\
= T(1) + N \cdot O(1) \\
= O(N)
\]

Apply recurrence again to \( T(N-1) \)

Total runtime

T(1) is base case, just \( O(1) \)
## Table of Recurrences

<table>
<thead>
<tr>
<th>Recurrence</th>
<th>Algorithm</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T(n) = T(n/2) + O(1)$</td>
<td>binary search</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>$T(n) = T(n-1) + O(1)$</td>
<td>sequential search</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>$T(n) = 2T(n/2) + O(1)$</td>
<td>tree traversal</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>$T(n) = T(n/2) + O(n)$</td>
<td>qsort partition, find $k^{th}$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>$T(n) = 2T(n/2) + O(n)$</td>
<td>mergesort, quicksort</td>
<td>$O(n \log n)$</td>
</tr>
<tr>
<td>$T(n) = T(n-1) + O(n)$</td>
<td>selection or bubble sort</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>

We expect you to be able to derive a recurrence relation from an algorithm, but not necessarily to solve. We will provide a table of solutions like this for exams.
Recurrence relation and runtime for traversing a balanced tree

- \( T(n) \) time \textbf{inOrder} (root) with \( n \) nodes
  - \( T(n) = T(n/2) + O(1) + T(n/2) = O(n) \)

- Why \( T(n/2) \)?

Assumes the tree is \textit{balanced}: Same number of nodes in the left subtree as the right.
Recurrence relation and runtime for traversing an unbalanced tree

- If every node has a right child but no left...
  - \( T(n) = T(0) + O(1) + T(n-1) = O(n) \)

So Tree traversal is \( O(n) \) regardless of balance.

What about search/contains and insert for a binary search tree?
Balance and runtime for search and insert in a binary search tree

If balanced?

- \( T(n) = T(n/2) + O(1) = O(\log(n)) \)

If unbalanced?

- \( T(n) = T(n-1) + O(1) = O(n) \)

```java
public boolean contains(TreeNode tree, String target) {
    if (tree == null) return false;
    int result = target.compareTo(tree.info);
    if (result == 0) return true;
    if (result < 0) return contains(tree.left, target);
    return contains(tree.right, target);
}
```
Why would a tree not be balanced?

Worse case:

• What if we insert sorted data?

For(int i=0; i<n; i++) {
    myTree.insert(i);
}

• Average case height $O(\log(n))$ for random-ish order

• AVL trees, red-black trees (later) can dynamically ensure good balance.
How much balance is enough?

**Approximate balance**: Say that a binary tree is 
\((a, b)\) —approximately balanced if...

• For every *node* rooting a subtree of size \(n \geq a\),
• The left and right subtrees of the *node* both contain at most \(b \left(\frac{n}{2}\right)\) nodes.

Then the recurrence relation for contains is...

• \(T(n) \leq T \left( b \frac{n}{2} \right) + O(1)\) for \(n \geq a\), and
• \(T(n) \leq T(n - 1) + O(1)\) for \(n < a\).
How much balance is enough?

So for example, if a binary tree is (5, 1.5)-approximately balanced...

Then the height of the tree is at most
\[ 5 + (\log_{2/1.5} n + 1) \]
\[ = 6 + \log_{4/3} n \]
\[ O(\log(n)) \]

Don’t need to worry about the derivation.

**Takeaway:** Approximate balance is good enough for good asymptotic performance.
HeightLabel APT

https://www2.cs.duke.edu/csed/newapt/heightlabel.html

• Create a new tree from a tree parameter
  • Same shape, nodes labeled with height
  • Use `new TreeNode`. With what values ...

Note that this APT 1-indexes height/depth. We introduced it 0-indexed and will otherwise stick to that convention.
FAQ: Can I make a tree?

```
public class TreeNode {
    int info;
    TreeNode left;
    TreeNode right;
    TreeNode(int x){
        info = x;
    }
    TreeNode(int x, TreeNode lNode, TreeNode rNode){
        info = x;
        left = lNode;
        right = rNode;
    }
}
```

Just call the TreeNode constructor for each new node and connect them.

```
TreeNode root = new TreeNode( x: 5);
root.left = new TreeNode( x: 3);
root.right = new TreeNode( x: 6);
root.left.left = new TreeNode( x: 2);
root.left.right = new TreeNode( x: 4);
```

More terse version

```
TreeNode myTree = new TreeNode( x: 5,
    new TreeNode( x: 3,
        new TreeNode( x: 2),
        new TreeNode( x: 4)),
    new TreeNode( x: 6));
```
Solving HeightLabel in Pictures

Base case: when null, 0

Recursive case: height of node is
1 + max(height of node.left
height of node.right)

When null? 0?

5 = 1 + max(4, 2)

2 = 1 + max(1, 1)

1 = 1 + max(0, 0)

10/31/22
Compsci 201, Fall 2022, L18: Tree Recursion
Solving HeightLabel in Code

```java
private int height(TreeNode t) {
    if (t == null) return 0;
    return 1 + Math.max(height(t.left),
                         height(t.right));
}

public class HeightLabel {
    public TreeNode rewire(TreeNode t) {
        // replace with working code
        return null;
    }
}

public TreeNode rewire(TreeNode t) {
    if (t == null) return null;
    return new TreeNode(height(t),
                         rewire(t.left),
                         rewire(t.right));
}
```

- **Base case:** when null, 0
- **Recursive case:** height of node is 1 + height of node.left + height of node.right
- **Method doesn’t just calculate height,** is supposed to create and return new tree with new nodes...
- **Using height helper method,** get height, create new node, return.
Tree Recursion tips / common mistakes

1. Draw it out! Trace your code on small examples.
2. Return type of the method. Do you need a helper method?
3. Base case first, otherwise infinite recursion / null pointer exception.
4. If you make a recursive call, make sure to use what it returns.
WOTO

Go to duke.is/cvp7b

Not graded for correctness, just participation.

Try to answer \textit{without} looking back at slides and notes.

But do talk to your neighbors!
Rewire runtime?

- recurrence of this all-green code? $T(n) =$
- $2T(n/2) + O(n)$
  - Balanced tree
    ```java
    public TreeNode rewire(TreeNode t) {
        if (t == null) return null;
        return new TreeNode(height(t),
                             rewire(t.left),
                             rewire(t.right));
    }
    ```
- $T(n-1) + O(n)$
  - Unbalanced
    ```java
    private int height(TreeNode t) {
        if (t == null) return 0;
        return 1 + Math.max(height(t.left),
                             height(t.right));
    }
    ```
HeightLabel Complexity

• Balanced? $O(N \log N)$,
  • $2T(n/2) + O(n)$

• Unbalanced, $O(N^2)$,
  • $T(N) = T(N-1) + O(N)$

• Do in $O(N)$ time? Yes, if we don't call height
  • Balanced: $T(N) = 2T(N/2) + O(1)$
  • Unbalanced: $T(N) = T(N-1) + O(1)$
HeightLabel in $O(N)$ time

- If recursion works, subtrees store heights!
- Balanced? $O(N)$,
  - $2T(n/2) + O(1)$
- Unbalanced, $O(N)$,
  - $T(N-1) + O(1)$

```java
public TreeNode rewire(TreeNode t) {
    if (t == null) { return null; }
    TreeNode leftOfMe = rewire(t.left);
    TreeNode rightOfMe = rewire(t.right);
    int lHeight = 0;
    int rHeight = 0;
    if (leftOfMe != null) { lHeight = leftOfMe.info; }
    if (rightOfMe != null) { rHeight = rightOfMe.info; }
    return new TreeNode(1+Math.max(lHeight, rHeight),
                        leftOfMe, rightOfMe);
}
```
Calculate the *diameter* of a binary tree, the length of the longest path (maybe through root, maybe not, can’t visit any node twice).