PFTFWoN

- Search trees, analysis, recurrences
  - Analyzing recursive methods typically means we'll use a recurrence equation/relation
  - Solve once, re-use solution 😊

- Writing Binary Tree code with APT system
  - Avoiding Practice-it, using the idea
  - How do input a tree? Output a tree?

- Applications of "trees"

Tree functions

- Compute height of a tree, what is complexity?
  - Length of longest root-to-leaf path

```java
int height(Tree root) {
    if (root == null) return 0;
    else {
        return 1 + Math.max(height(root.left),
                             height(root.right));
    }
}
```

- Modify function to compute number of nodes in a tree, does complexity change?
  - What about computing number of leaf nodes?

Balanced Trees and Complexity

- A tree is height-balanced if
  - Left and right subtrees are height-balanced
  - Left and right heights differ by at most one

```java
boolean isBalanced(Tree root){
    if (root == null) return true;
    return isBalanced(root.left) && isBalanced(root.right) &&
           Math.abs(height(root.left) - height(root.right)) <= 1;
}
```
What is complexity?

- Assume trees “balanced” in analyzing complexity
  - Roughly half the nodes in each subtree
  - Leads to easier analysis

- How to develop recurrence relation?
  - What is T(n)? Time `func` executes on n-node tree
  - What other work? Express recurrence, solve it

- How to solve recurrence relation
  - Plug, expand, plug, expand, find pattern
  - Proof requires induction to verify correctness

Recurrence relation

- Let T(n) be time for `height` to execute (n-node tree)
  - T(n) = T(n/2) + T(n/2) + O(1)
  - T(n) = 2T(n/2) + 1
  - T(n) = 2[2T(n/4) + 1] + 1
  - T(n) = 4T(n/4) + 2 + 1
  - T(n) = 8T(n/8) + 4 + 2 + 1, eureka!
  - T(n) = 2^kT(n/2^k) + 2^k - 1 why is this true?
  - T(n) = nT(1) + O(n) is O(n), if we let n=2^k

- Let T(n) be time for `isBalanced` on n-node tree
  - T(n) = 2T(n/2) + O(n), why? Solution?

Recurrence Summary

<table>
<thead>
<tr>
<th>Recurrence</th>
<th>Algorithm</th>
<th>complexity</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) = 2T(n/2) + O(n)</td>
<td>Quicksort</td>
<td>O(n log n)</td>
</tr>
<tr>
<td>T(n) = T(n-1) + O(n)</td>
<td>Selection Sort</td>
<td>O(n^2)</td>
</tr>
<tr>
<td>T(n) = 2T(n-1) + O(1)</td>
<td>Towers of Hanoi</td>
<td>O(2^n)</td>
</tr>
</tbody>
</table>

- T(n): time for `isBalanced` to execute (n-node tree)
  - So, solution for T(n) = 2T(n/2) + O(n)
    - O(n log n) – base 2, but base doesn't matter

- Typically we say T(1) is O(1) in solving recurrence.
Doubly Linked Lists

- Why do we have some lists with nodes to previous and next nodes?
  - Easier to write code, don't need before, current, after ...
  - See examples in Recitation
- Used in Java through Java 7 for HashMap
- Still used in LinkedList, easy traversal from front or back, also LinkedHashSet
  - See DNA LinkStrand with singly linked list

LinkedHashMap in code

- Note that each hash "bucket" uses a search tree to store (key,value) pairs where keys have same hashcode
  - Search tree nodes linked using doubly-linked list with before and after pointers

WOTO

http://bit.ly/201fall16-nov4-1

We'll review solutions to recurrences, but this is about constructing the recurrences
Lynn Conway

See Wikipedia and lynnconway.com

- Joined Xerox Parc in 1973
  - Revolutionized VLSI design with Carver Mead
- Joined U. Michigan 1985
  - Professor and Dean, retired ’98
- NAE ’89, IEEE Pioneer ’09
- Helped invent dynamic scheduling early ’60s IBM
  - Transgender, fired in ’68

What does insertion look like?

- Simple recursive insertion into tree (accessed by root)
  
  ```java
  root = insert("foo", root);
  ```

  ```java
  TreeNode insert(TreeNode t, String s) {
    if (t == null) t = new Tree(s,null,null);
    else if (s.compareTo(t.info) <= 0)
      t.left = insert(t.left,s);
    else
      t.right = insert(t.right,s);
    return t;
  }
  ```

Notes on tree insert and search

- Note: in each recursive `insert` call, the parameter `t` in the called clone is either the left or right pointer of some node in the original tree
  - Why is this important?
  - The idiom `t = treeMethod(t,..)` used
- When good trees go bad, what happens and why?
  - Insert alpha, beta, gamma, delta, epsilon, ...
  - Where does gamma go?
    - Can we avoid this case? Yes!
    - What to prefer? Long/stringy or short/bushy
Removal from tree?

- For insertion we can use iteration (see BSTSet)
  - Look below, either left or right
    - If null, stop and add
    - Otherwise go left when <=, else go right when >

- Removal is tricky, depends on number of children
  - Straightforward when zero or one child
  - Complicated when two children, find successor
    - See set code for complete cases
    - If right child, straightforward
    - Otherwise find node that's left child of its parent (why?)

Balanced Trees

- In average case, trees are O(log n) for searching in a binary search tree
  - If tree is full (see picture) or complete (last row being filled in) we get O(log n)
- Insertion/Deletion algorithms that "adjust"
  - Ensure trees good in worst-case
- AVL tree, Red-Black Tree
  - Similar, Red-Black in TreeMap
- 2-3 Trees or 2-3-4 Trees
  - Leaves in file system?

Writing Tree, Serializing Tree

- Conventional wisdom: need two traversals to uniquely identify tree
  - https://en.wikipedia.org/wiki/Tree_traversal

Given Inorder and Preorder

- Suppose we have a search tree, we know in-order
  - Are there more trees with these values?
  - Are these search trees?
Construct tree given pre-order?

- Given 7, 4, 1, 3, 6, 13, 8, 10, 14
  - Knowing it’s a search tree: what’s root? Left child of root? Right child of root?

- If we implement the algorithm developed, what will complexity be? How do we reason about this?
  - T(n) = ...
  - What are sub-problems?
  - What work before we do subproblems?

Label null nodes, pre-order is unique

- What is the tree (not a search tree) with pre-order 5, 9, x, 4, 7, x, x, 2, 8, x, 3, x, x, 6, 1, x, x, 10, x, x
  - What is root? Why?
  - What's the left-subtree, how do we "stop" scanning the traversal?

- Given a tree of N nodes, how many x’s are there?
  - How do reason about this?
  - Is this traversal "wasteful" in storing nulls?

Solving APTs with Trees, aka TAPT

- Using a TreeNode class in the code you write

- Let's work on some of these to see how the process works, then do more in recitation as a way of preparing for both AutoComplete and Huffman coding and for tests

- Write by hand, debug with your mind, test with your computer