Data Structures: Trees
Learning Objectives

• Trees are a general version of linked lists
• Trees have a value, and are connected to “sub-trees” called branches
• We can often use recursion to process all items in a tree
  – We typically have recursion inside a loop over all the tree’s branches
  – This is called “Depth First Search”
Why Use Trees?

• Trees represent lots of natural structures
  – A boss who has employees report to them
  – Courses which belong to departments, and departments which colleges in a University
  – Anything with a hierarchy, really.
    » A family tree
    » Biological taxonomies (Kingdom, Phylum....)
    » Files and Folders
Review: Linked Lists

• A Recursive List, sometimes called a "rlist"
• Linked lists contain other linked lists
• A series of items with two pieces:
  – A value, usually called "first"
  – A “pointer” to the rest of the items in the list.

We’ll use a very small Python class “Link” to model this.
What is a tree?

• A recursive data structure
  – Almost like a linked list!
• What if a linked list could have multiple “rest” element
• We call these “branches”.
• Each branch is also its own Tree.
Trees are common in Computer Science

• Trees give us really cool approaches for “divide and conquer”
  – Used in every computer to speed up searching for files
  – Used for modeling decision systems in AI programs
  – Used for modelling the kinds of moves in a game.
• Another recursive data structure!
  – We can keep practicing recursion and working with classes
  – Computer sciences really like recursion. 😊
• Trees are a simplified form of a graph, a tool which can help us model just about anything.
Computational Structures in Data Science

Trees: Code Overview
(Go Inspect the ipynb)
class Tree:
    def __init__(self, value, branches=[]):
        self.value = value
        for branch in branches:
            assert isinstance(branch, Tree)
        self.branches = list(branches)
    def __repr__(self):
        if self.branches:
            branches_str = ', ' + repr(self.branches)
        else:
            branches_str = ''
        return 'Tree({0}{1})'.format(self.value, branches_str)
    def is_leaf(self):
        return not self.branches
    def add_branch(self, tree):
        assert isinstance(tree, Tree), "Each branch of a Tree must be an instance of a Tree"
        self.branches.append(tree)
Trees:
Practice With Recursion:
traverse_recursive
Trees:
Counting Each Node
How do we count nodes?

- The “root” or top of the tree is one node.
  - (We assume we can’t have a tree of 0 nodes!)
- For each subtree we... Count the nodes!
  - Doesn’t this sound like recursion?
- Trick: How do we group the results of recursion?
def count_nodes(t):
    """The number of leaves in tree.

    >>> count_nodes(fib_tree(5))
    8
    ""
    if t.is_leaf():
        return 1
    else:
        return 1 + sum(map(count_nodes, t.branches))
Trees:
Practice With Recursion:
print_tree
Trees:
Advanced Topics: Searching
Optional!
Searching Trees: Two Strategies

• The searching we have been doing today is called “Depth First Search”, or DFS.
• Recursion makes the algorithm very nice.
  – First: we deal with our current item, then we get to the branches.
  – We always make a recursive call on the first branch
  – We continue recursing until there are no more branches
  – Then the function executes, and we go back “up” a level and check out the next branch.
  – We sometimes say: “popping up the stack”.
• The stack is the “stack of function calls” the computer uses to keep track of how things work, and you’ll learn about this in CS61B.
Searching a Tree by level: Breadth First Search

• What if I want to check out all the values of my branches before making a recursive call?
• What if we said, you just can’t use recursion. (Sometimes, CS instructors do weird things like that...)
• This is used in practice for lots of cool things:
  – Shortest path between two items (more of a graph and not a tree, usually). Google Maps uses it for routing and the algorithms that power the internet use it.