Computational Structures in Data Science

Data Structures:
Trees

Berkeley
UNIVERSITY OF CALIFORNIA

Announcements – Climate / Culture

- The rumors of my demise are greatly exaggerated. ☺
- Reminder: Climate and culture in the classroom are important.
  - https://eecs.link/climate EECS Climate Form
  - CDSS / Data Science climate form
  - You are free to use either or both forms.
- Midterm Grading:
  - Regrade Requests due tonight!
  - Will resolve them by end up the week
  - Grading updates next week.
    - Likely dropping the lowest 2-3 questions / scaling scores
    - Will release grade reports soon.

Announcements

- Ants Project out next week
  - ~ 3 weeks long
  - **Partners recommended, but work *together!***
  - **Do not "trade off" questions!**

Learning Objectives

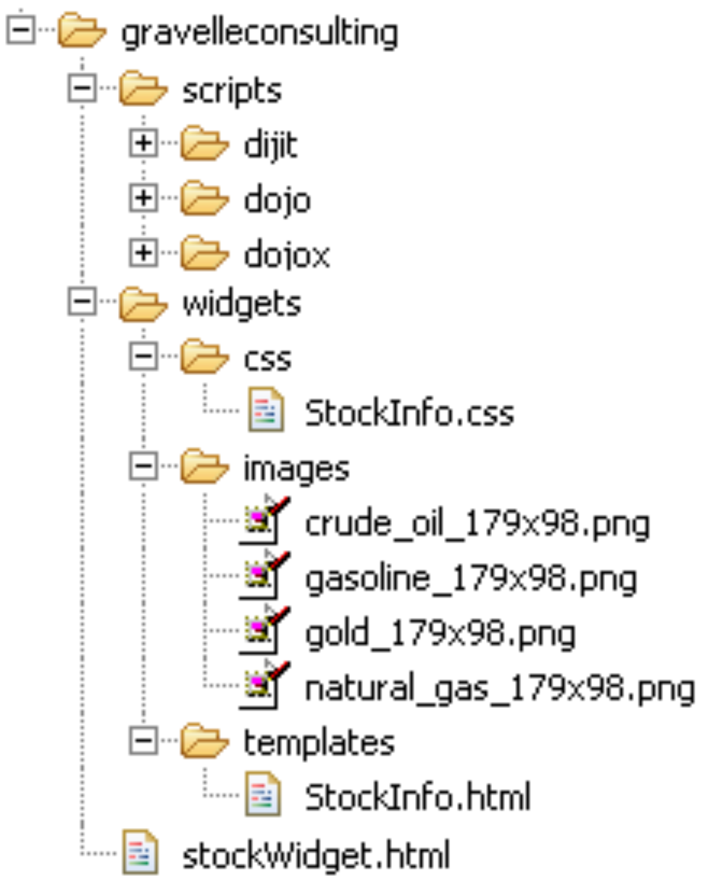
- Trees can be seen as a general version of linked lists
- Trees have a value, and are connected to "sub-trees" called branches
- We can often use recursion to process all items in a tree
  - We typically have recursion inside a loop over all the tree's branches
  - This is called "Depth First Search"

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
  - A game board, which representers as series of potential moves, one after the next.
    - E.g. Given Move 1, link the next possible moves, which link the next set of possible moves

Recall: Tree Recursion

List all items on your hard disk



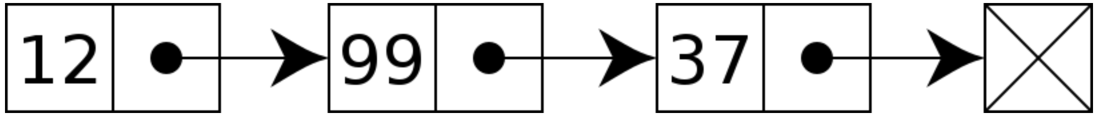
- Files
- Folders contain
  - Files
  - Folders

```
def process_directory(directory):
    for item in directory:
        if is_file(item):
            process_file(item)
        else:
            process_directory(item)
```

Review: Linked Lists

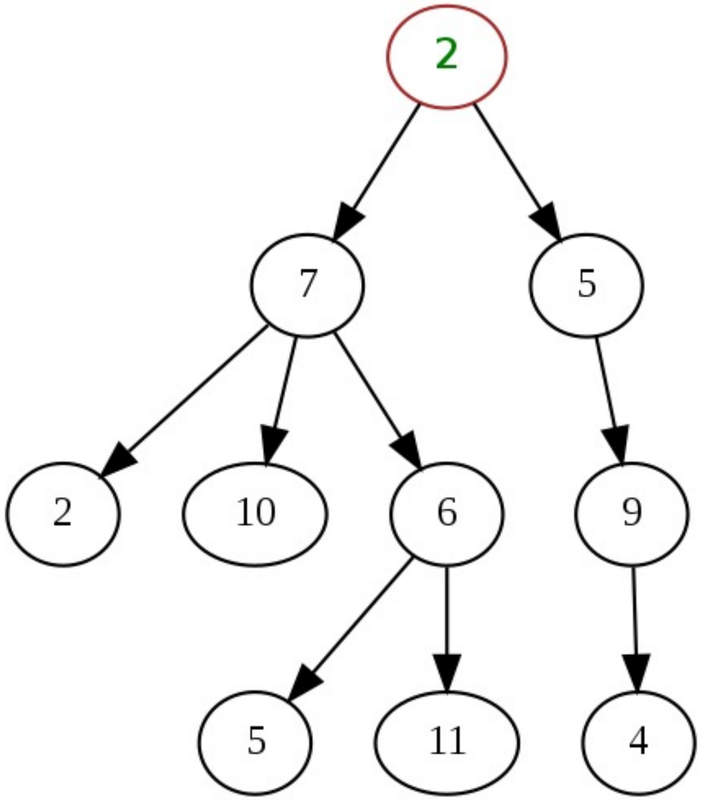
- A Recursive List, sometimes called a "rlist"
- Linked lists contain other linked lists
- A series of items with two pieces:
  - A value, usually called `"first"`
  - A “pointer” to the `rest` of the items in the list.

- We’ll use a very small Python class “Link” to model this.



What is a tree?

- **What is a tree?** (in CS...)
  - A *data structure* -- an organization of objects in a particular format.
  - In this case, Objects which have one or more *children*
- Almost like a linked list!
  - What if a linked list could have multiple "rest" elements?
- We call these children elements *branches*.

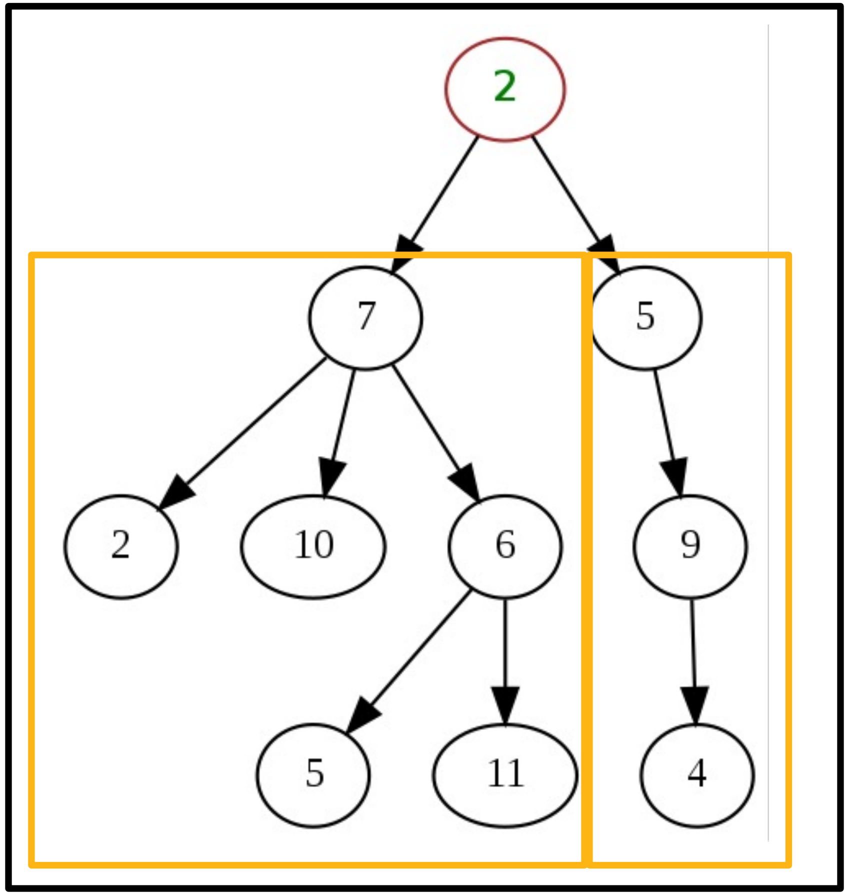


Trees are Recursive

- Another recursive data structure!
  - We can keep practicing recursion and working with classes
  - Computer science really likes recursion. ☺
- Recall: *tree recursion* from before the midterm...
  - **Tree recursion exists independently of tree data structures!**
  - However, tree recursion is a common technique for processing data in trees.
- **Each branch is also its own Tree.**

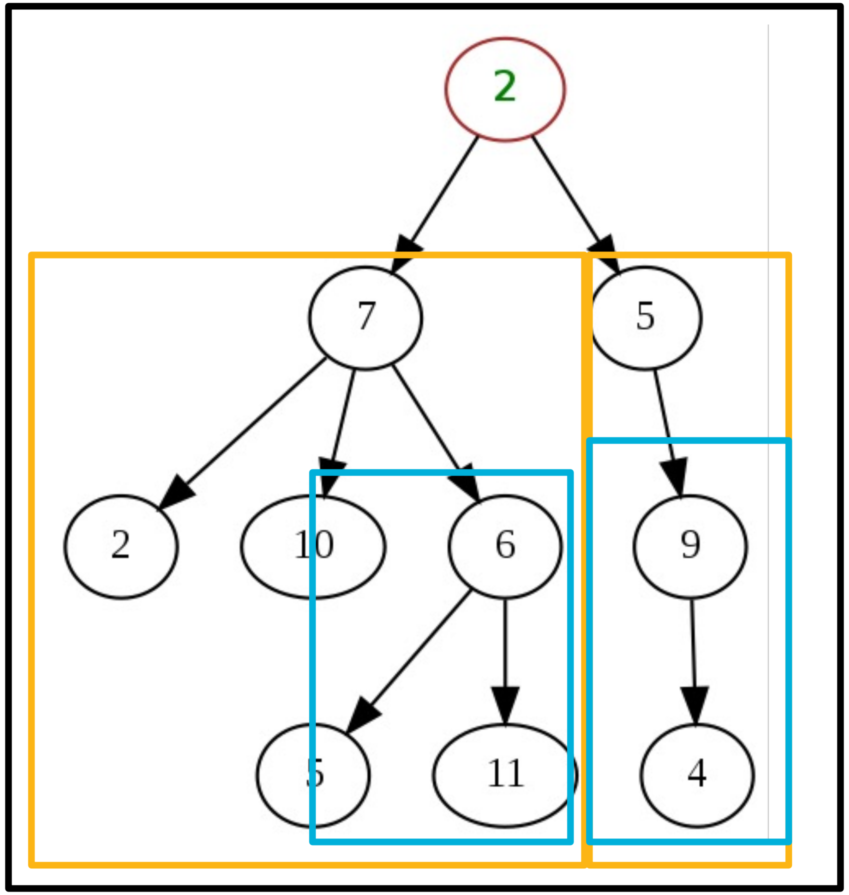
How many trees are there?

- Mostly a rhetorical question…



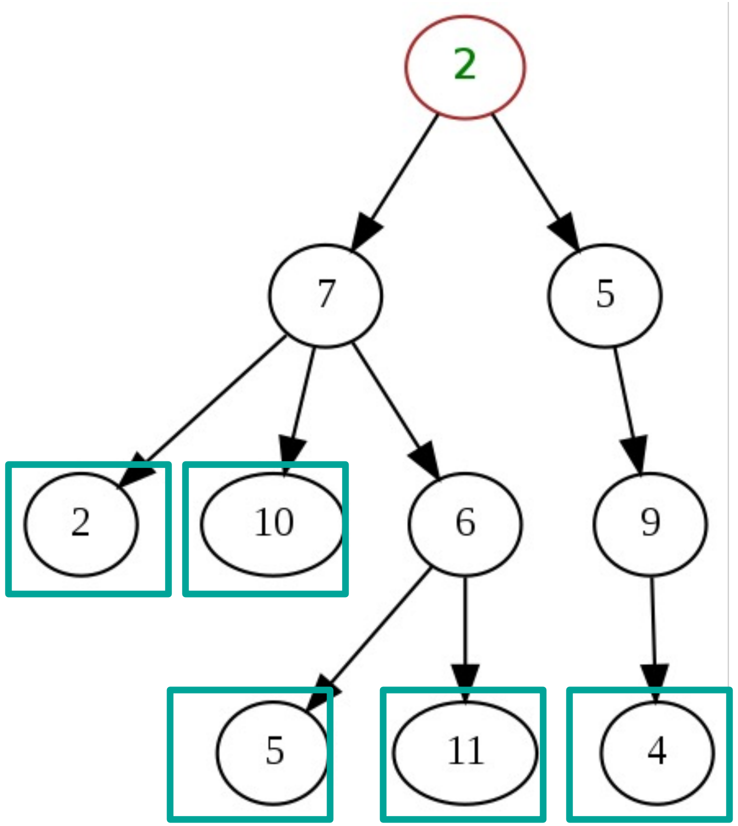
How many trees are there?

- Mostly a rhetorical question...



How many trees are there?

- Leaf notes are the elements with no branches
- They are *also* a Tree – this will make our recursive code simpler. ☺



Trees are common in Computer Science

- We shown trees showing numbers *for simplicity.*
  - In practice, the value of each tree element varies a lot!
  - In can be a person, a course, any kind of object.
- Trees give us awesome approaches for “divide and conquer”
  - Used in every computer to speed up searching for files (Binary search!)
  - Used for modeling decision systems in AI programs
  - Used for modelling the potential moves in a game.
- Trees are a simplified form of a *graph*, a tool which can help us model just about anything.
  - Graphs are a (relatively) important topic in CS61B

Computational Structures in Data Science

Trees: Code Overview

(Go Inspect the ipynb)

Berkeley
UNIVERSITY OF CALIFORNIA

What’s a tree? (C88C-style)

- A tree is a list of trees!
- Each tree has a node, with a value.
- Each node has `branches` which are itself, trees.
  - There can be zero or many branches
- There is always 1 ”root” node

Our Simple Tree Class: A couple new methods!

```
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):
        branches_str = ''
        if self.branches:
            branches_str = ', ' + repr(self.branches)
        return f'Tree({self.value}{braches_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)
```

Computational Structures in Data Science

Trees:
Practice With Recursion:
`traverse_recursive`

Berkeley
UNIVERSITY OF CALIFORNIA

Computational Structures in Data Science

Trees:
Counting Each Node

Berkeley
UNIVERSITY OF CALIFORNIA

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?
- Hard Part: How do we group the results of recursion?
- Remember our recursive algorithm:
  - Base case
  - Recursive Case:
    - Divide
    - Invoke
    - Combine

```
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))
```

Computational Structures in Data Science

Trees:
Practice With Recursion:
`print_tree`

Berkeley
UNIVERSITY OF CALIFORNIA

Computational Structures in Data Science

Trees:
Advanced Topics: Searching
Optional!

Berkeley
UNIVERSITY OF CALIFORNIA

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.