CS 61A SU26

Lecture 09: Trees

July 6, 2026

Rebecca Dang

Join pollev.com/rebeccadang831 (or scan QR code) on your phone or laptop

We'll begin at Berkeley time (10 minutes after), as per tradition!

Potpourri

5 min

- Announcements
- Lecture 08 Follow-Up
- Computing in the News

Announcements

- Quiz 2 happening today and tomorrow
- Upon request, I've added the **PollEv full question text and answer options** in speaker notes 🎉
  - Alternatively, watch recording on bCourses
- **New OH hours and locations**, check the OH calendar (for all staff + instructor)
  - Ed post
  - Instructor OH:
    - Tue 1-2 pm moved to **Cory 540AB starting tomorrow**
    - Thu 7-8 pm moved to **Cory 521 starting next week** (still in Soda 347 this week)
  - Instructor Tea Hours: Wed 11:30 am - 12:30 pm still Soda 795 this week, but **subject to change starting next week** (hoping for Cory room)
- **Additional Exam Prep** sections available
  - **Caution**: Read the Ed post to see when the new sections start
- **Student support** meetings available
- No post-lecture questions today

[out of scope] Lecture 08 Follow-Up

Python dictionary implementation

- TLDR: it's just a hash table with open addressing and some space optimizations
- Blogs in increasing level of detail; they may be outdated...
  - How are Python's Built In Dictionaries Implemented? (StackOverflow)
  - Python dictionary, the implementation
  - Python behind the scenes #10: how Python dictionaries work
- More in CS 61B!

Computing in the News

- ‘Who Should I Vote for?’ Voters Turn to A.I. Before Casting Their Ballots (New York Times)
- UC Berkeley shows no signs of AI grade inflation as professors adapt (Daily Cal)

Review

10 min

- Dictionaries
- ADTs

Practice: Flip Dict

- Implement `flip_dict`
  - Hint: To check if something is a particular data type in Python, you can do:
    - `type(<expression>) is <type>` or
    - `type(<expression>) == <type>` (less preferred) or
    - `isinstance(<expression>, <type>)`
- Download starter code `09.py` from the course website under Lecture 9
- Run doctests with `python3 -m doctest 09.py`
- 5 min: Try implementing it yourself
- 5 min: Discuss with someone next to you and compare your implementations
  - What worked?
  - What didn't?
  - Why?

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

Consider the Coordinate ADT we implemented in Lecture 08. Which of the following functions are behind the abstraction barrier (e.g. part of the ADT)?

Trees

15 min

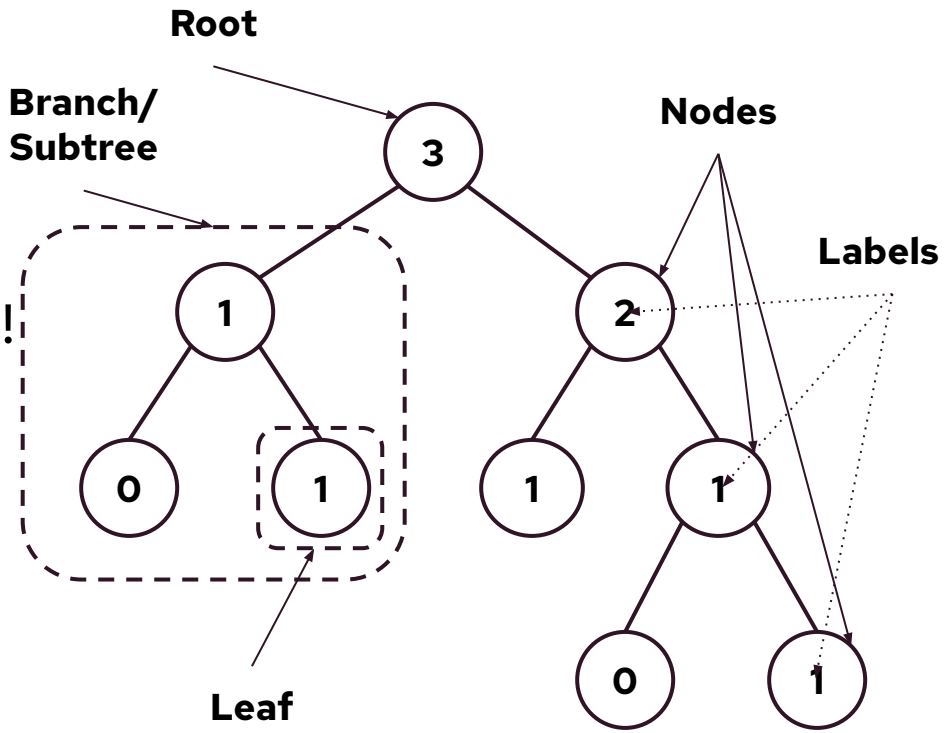
- Definition
- Motivation
- Tree Terminology
- Tree ADT

Definition (1 of 2)

A **tree** is a data structure comprised of a **root** node with a **label** and **branches**, which is a sequence of **(sub) trees**.

Note: In CS, trees are upside-down!

(More formal definition in CS 70, and even more trees in CS 61B!)



Definition (2 of 2)

Recursive

A tree has a root node and its branches are also trees (subtrees)

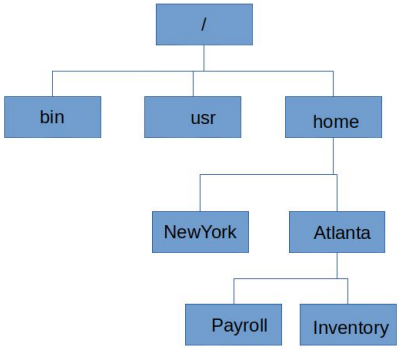
Relative

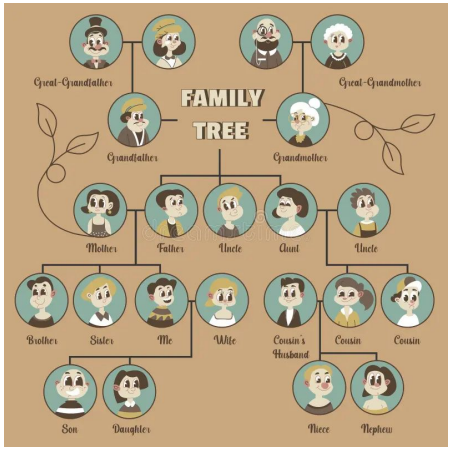
A node has a **parent** and 0 or more **children**

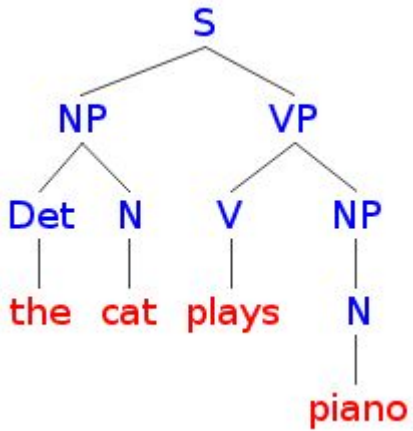
Motivation

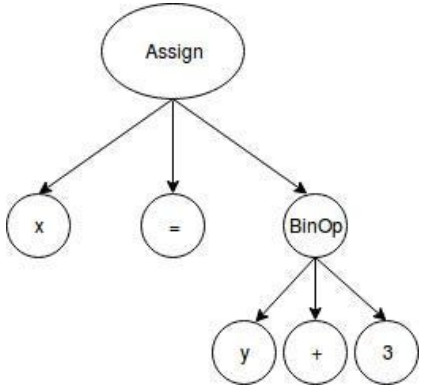
Why do we care about trees? They represent a lot of real-world structures!

- File system
- Hierarchy (political, social, organizational, etc.)
- Family lineages
- Syntax trees (in linguistics (LING 100) and computer science (CS 164))



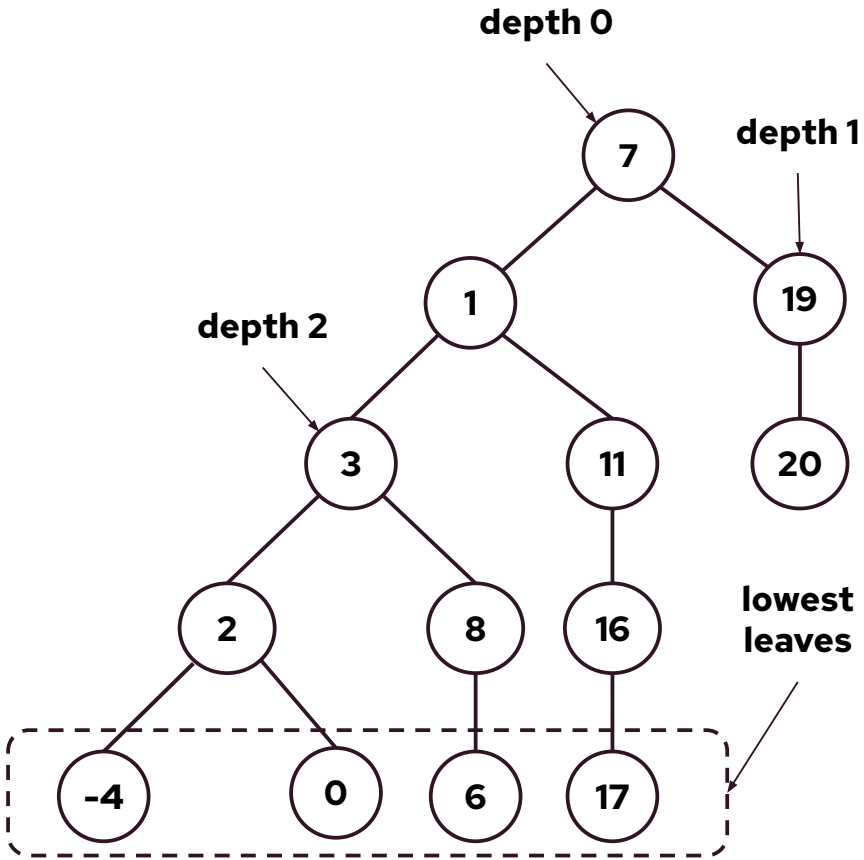






Tree Terminology

- **Depth**: How far away a node is from the root.
  - In other words, the number of edges between the root of the tree to the node.
  - Property of a node
- **Height:** The depth of the lowest leaf.
  - Property of a tree



Tree ADT (1 of 3)

```
def tree(root_label, branches=[]):
    for branch in branches:
        assert is_tree(branch), 'branches must be trees'
    return [root_label] + list(branches)

def label(tree):
    return tree[0]

def branches(tree):
    return tree[1:]
```

Tree ADT (2 of 3)

```
def is_tree(tree):
    if type(tree) != list or len(tree) < 1:
        return False
    for branch in branches(tree):
        if not is_tree(branch):
            return False
    return True

def is_leaf(tree):
    return not branches(tree)
```

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

Suppose we have already defined the tree ADT as mentioned in the lecture slides. What would Python display if we evaluated tree(5, [1])?

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

Suppose we have already defined the tree ADT as mentioned in the lecture slides. True or false: The following expression is an abstraction barrier violation: tree(1, [tree(2)])

Tree ADT (3 of 3)

```
>>> t = tree(3, [tree(1), tree(2, [tree(1), tree(1)])])
>>> t
[3, [1], [2, [1], [1]]]
>>> label(t)
3
>>> branches(t)
[[1], [2, [1], [1]]]
>>> label(branches(t)[1])
2
>>> is_leaf(t)
False
>>> is_leaf(branches(t)[0])
True
```

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

Which of the following tree diagrams corresponds to this expression? tree(5, [tree(1, [tree(0)]), tree(3, [tree(4, [tree(6)])]), tree(2)])

Indentation helps readability!

```
tree(5, [tree(1, [tree(0)]), tree(3, [tree(4, [tree(6)])]), tree(2)])
```

```
tree(5, [tree(1, [tree(0)]),
         tree(3, [tree(4, [tree(6)])]),
         tree(2)])
```

Tree Processing

15 min

- Fibonacci Tree
- Count Leaves

Fibonacci Tree

```
def fib_tree(n):
    """
    Returns a tree representation of the nth Fibonacci number
    """
    if n == 0 or n == 1:
        return tree(n)
    else:
        left, right = fib_tree(n - 2), fib_tree(n - 1)
        fib_n = label(left) + label(right)
        return tree(fib_n, [left, right])
```

code.cs61a.org

1. Go to code.cs61a.org and copy/paste the `fib_tree` definition into a new Python file.
2. Click the green play button in the top right
3. Let's use the `autodraw()` feature to see `fib_tree(5)` in action!

Count Leaves

```
def count_leaves(t):
    """Returns the number of leaves in the tree t"""
    if is_leaf(t):
        return 1
    else:
        branch_counts = [count_leaves(b) for b in branches(t)]
        return sum(branch_counts)
```

5 min break

Practice

(reminder to self to use high contrast theme)

30 min

- Leaves
- Double
- Count Paths

Practice: Leaves

- Implement `leaves`
  - Hint: If you `sum` a list of lists, you will get a list containing the elements of those lists, e.g. `sum([ [1], [2, 3], [4] ], [])` is `[1, 2, 3, 4]`
- Download starter code `09.py` from the course website under Lecture 9
- Run doctests with `python3 -m doctest 09.py`
- 5 min: Try implementing it yourself
- 5 min: Discuss with someone next to you and compare your implementations
  - What worked?
  - What didn't?
  - Why?

Practice: Double

- Implement `double`
- Download starter code `09.py` from the course website under Lecture 9
- Run doctests with `python3 -m doctest 09.py`
- 5 min: Try implementing it yourself
- 5 min: Discuss with someone next to you and compare your implementations
  - What worked?
  - What didn't?
  - Why?

Practice: Count Paths

- Implement `count_paths`
  - Hint: Read the doctests carefully!
  - Hint: The tree `t` in the doctests is pictured to the right
- Download starter code `09.py` from the course website under Lecture 9
- Run doctests with `python3 -m doctest 09.py`
- 5 min: Try implementing it yourself
- 5 min: Discuss with someone next to you and compare your implementations
  - What worked?
  - What didn't?
  - Why?