Trees

Announcements

A quick aside…

On Exam Scores

My instructor OH this week will be extended and will focus on exam support sessions - other staffers will be helping out. We can get a sense of your studying style and give advice headed into the final! Keep an eye out on EdStem.

Partly because the format of the exam did not match past exams, but now you know what to expect for the final ;)

* A low midterm score does not need to define who you are. It is OK to feel behind, disheartened, exhausted. Take care of yourself and your wellbeing, *then* turn your attention back to C88C.

* We are here to support you! OH, EdStem, exam prep tutoring

* Midterm 2 12.0 / 45.0 Midterm 2 29.5 / 60.0

^— a few of my exam scores as an undergrad

Data Abstraction

Data Abstraction

A small set of functions enforce an abstraction barrier between ***representation*** and ***use***

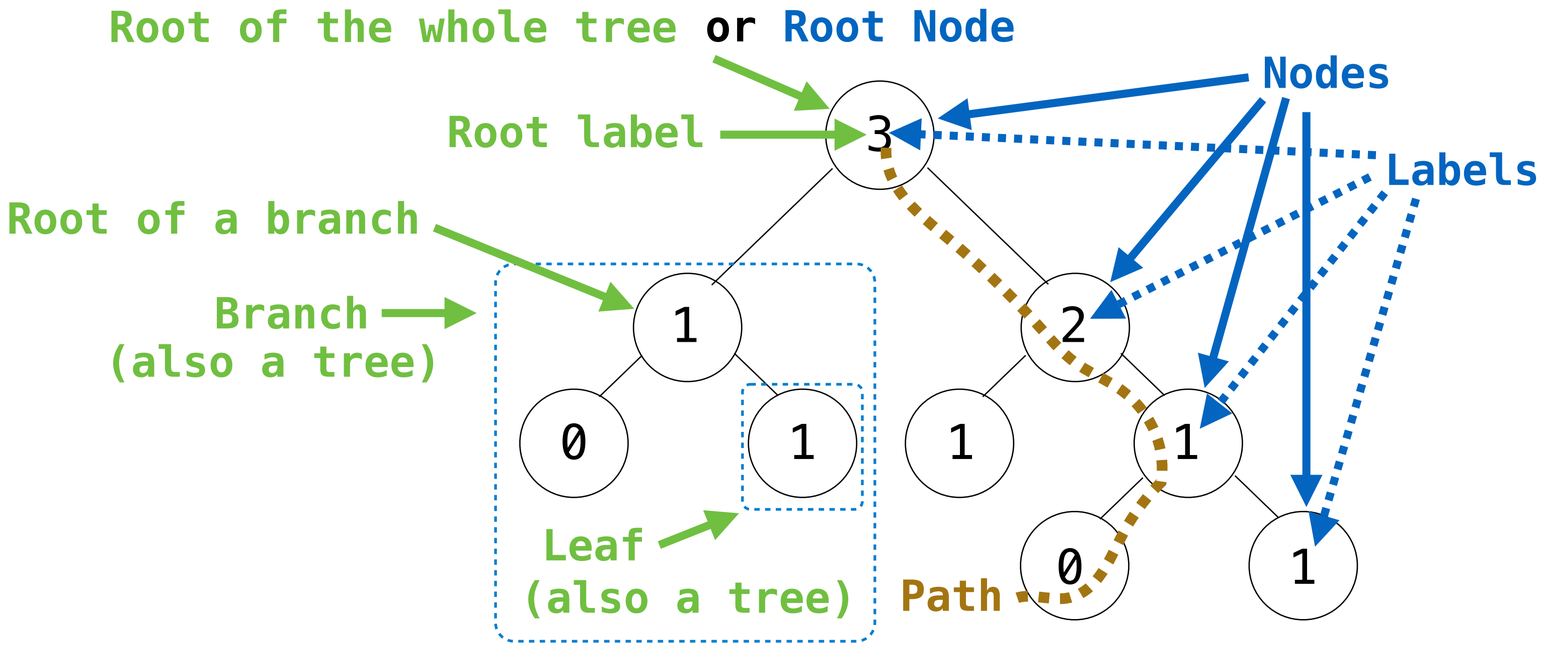
- How data are represented (as some underlying list, dictionary, etc.)
- How data are manipulated (as whole values with named parts)

E.g., use Link.empty instead of ()

Why? Code becomes easier to read & revise; later you could represent Link.empty as None or [] or “”

Trees

Tree Abstraction



Recursive description (wooden trees):

A **tree** has a **root label** and a list of **branches**

Each **branch** is a **tree**

A **tree** with zero **branches** is called a **leaf**

A **tree** starts at the **root**

Relative description (family trees):

Each location in a tree is called a **node**

Each **node** has a **label** that can be any value

One node can be the **parent/child** of another

The top node is the **root node**

People often refer to labels by their locations: "*each parent is the sum of its children*"

Using the Tree Abstraction

For a tree t, you can **only**:

- Get the label for the root of the tree: `t.label`
- Get the list of branches for the tree: `t.branches`
- Get the branch at index i, which is a tree: `t.branches[i]`
- Determine whether the tree is a leaf: `t.is_leaf()`
- Treat t as a value: `return t`, `f(t)`, `[t]`, `s = t`, etc.

(Demo)

Tree Processing

Tree Processing Uses Recursion

Processing a leaf is often the base case of a tree processing function

The recursive case typically makes a recursive call on each branch, then aggregates

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

Writing Recursive Functions

Make sure you can answer the following before you start writing code:

- What recursive calls will you make?
- What type of values do they return?
- What do the possible return values mean?
- How can you use those return values to complete your implementation?

Practice: Print_Sums

Processing a leaf is often the base case of a tree processing function

The recursive case typically makes a recursive call on each branch, then aggregates

```
    def print_sums(t):
        """Print the sum of labels along the path from the root to each leaf.
        >>> print_sums(tree(3, [tree(4), tree(5, [tree(6)])]))
        7
        14
        """
```

Practice: print_sums_helper

```
def print_sums_helper(t, path_sum):
    """Print the sum of labels along the path from the root to each leaf.
    >>> print_sums_helper(tree(3, [tree(4), tree(5, [tree(6)])]), 0)
    7
    14
    """
    path_sum = ____________
    if ____________________:
        print(path_sum)
    else:
        for branch in t.branches:
            ______________________
```

Practice: print_sums_helper

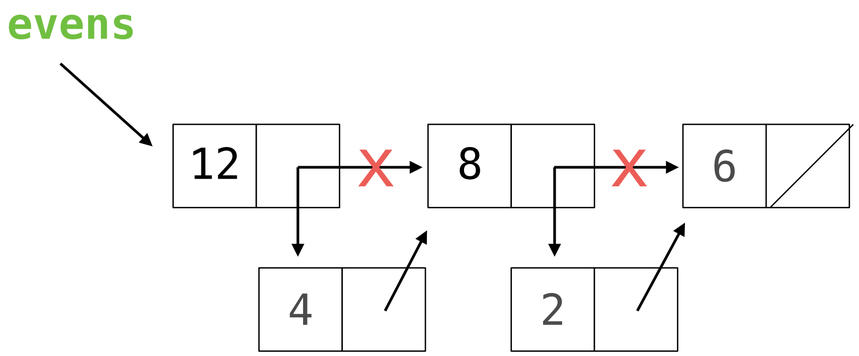
```
def print_sums_helper(t, path_sum):
    """Print the sum of labels along the path from the root to each leaf.
    >>> print_sums_helper(tree(3, [tree(4), tree(5, [tree(6)])]), 0)
    7
    14
    """
    path_sum = path_sum + t.label
    if t.is_leaf():
        print(path_sum)
    else:
        for branch in t.branches:
            print_sums_helper(branch, path_sum)
```

Practice: Linked Lists

Inserting into a Linked List

```
def insert_link(s, x, i):
    """Insert x into linked list s at index i.

    >>> evens = Link(4, Link(2, Link(6)))
    >>> insert_link(evens, 8, 1)
    >>> insert_link(evens, 10, 4)
    Index out of range
    >>> insert_link(evens, 12, 0)
    >>> insert_link(evens, 14, 10)
    Index out of range
    >>> print(evens)
    <12 4 8 2 6>
    """
    if s is Link.empty:
        print('Index out of range')
    elif i == 0:
        second = ____________________
        s.first = _______________
        s.rest = second
    else:
        insert_link(s.rest, x, i-1)
```



Inserting into a Linked List

```
def insert_link(s, x, i):
    """Insert x into linked list s at index i.

    >>> evens = Link(4, Link(2, Link(6)))
    >>> insert_link(evens, 8, 1)
    >>> insert_link(evens, 10, 4)
    Index out of range
    >>> insert_link(evens, 12, 0)
    >>> insert_link(evens, 14, 10)
    Index out of range
    >>> print(evens)
    <12 4 8 2 6>
    """
    if s is Link.empty:
        print('Index out of range')
    elif i == 0:
        second = Link(s.first, s.rest)
        s.first = x
        s.rest = second
    else:
        insert_link(s.rest, x, i-1)
```