Lecture #21: Tree Structures

A General Tree Type

- Trees don't quite lend themselves to being captured with standard syntax like tuples or lists, because they get accessed in various ways, with slightly varying interfaces.
- For the purposes of this lecture, we'll use this type, which has no empty trees:

```
class Tree:
    """A Tree consists of a label and a sequence
    of 0 or more Trees, called its children."""

    def __init__(self, label, *children):
        """A Tree with given label and children.
        For convenience, if children[k] is not a Tree,
        it is converted into a leaf whose operator is
        children[k]."""
        self.__label = label;
        self.__children = \
          [ c if type(c) is Tree else Tree(c)
              for c in children]
```

A General Tree Type: Accessors

```
# class Tree:
    @property
    def is_leaf(self):
        return self.arity == 0

    @property
    def label(self):
        return self.__label

    @property
    def arity(self):
        """The number of my children."""
        return len(self.__children)

    def __iter__(self):
        """An iterator over my children."""
        return iter(self.__children)

    def __getitem__(self, k):
        """My kth child."""
        return self.__children[k]
```

A Simple Recursion

- Since trees are recursively defined, recursion generally figures in algorithms on them.
- Example: number of leaf nodes.

```
def leaf_count(T):
    """Number of leaf nodes in the Tree T."""
    if T.is_leaf:
        return 1
    else:
        s = 0
        for child in T:
            s += leaf_count(child)
        return s
        # Can you put the else clause in one line instead?
        return ?
```

- How long does this take (for a tree with N leaves)?

A Search

- This particular definition of trees lends itself to Noetherian induction with no explicit base case.

```
def tree_contains(T, x):
    """True iff x is a label in T."""
```

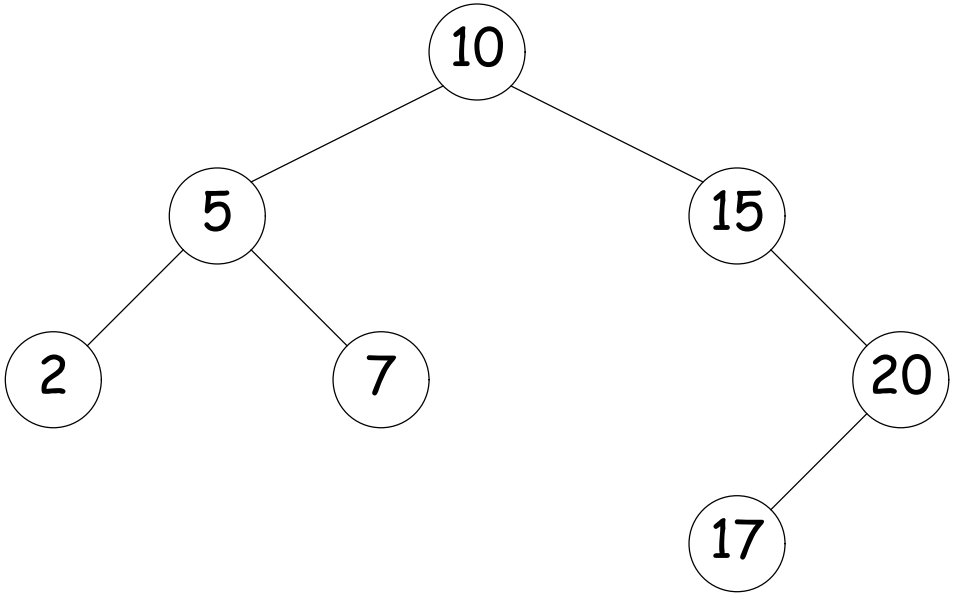
Tree to List

- Another example with no explicit base cases:

```
def tree_to_list_preorder(T):
    """The list of all labels in T, listing the labels
    of trees before those of their children, and listing their
    children left to right (preorder)."""
```

Ordered Trees

- The book talks about *search trees* as implementations of sets of values.
- Here, the purpose of the tree is to divide data into smaller parts.
- In a *binary search tree,* each node is either empty or has two children that are binary search trees such that all labels in the first (left) child are less than the node's label and all labels in the second (right) child are greater.



Tree Search Program

```
def tree_find(T, x):
    """True iff x is a label in set T, represented as a search tree.
    That is, T
       (a) Represents an empty tree if its label is None, or
       (b) has two children, both search trees, and all labels in
           T[0] are less than T.label, and all labels in T[1] are
           greater than T.label."""
```