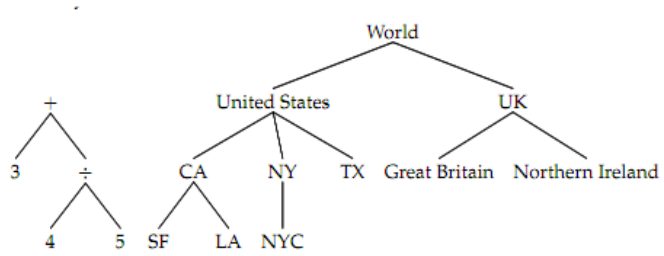
CS61A Lab 9: Recursive Data - Trees and Sets

Week 9, 2012

General Trees (with infinite children)

Trees are a way we have of representing a hierarchy of information. A family tree is a good example of something with a tree structure. You have a matriarch and a patriarch followed by all the descendants. Alternately, we may want to organize a series of information geographically. At the very top, we have the world, but below that we have countries, then states, then cities. We can also decompose arithmetic operations into something much the same way.



The name “tree” comes from the branching structure of the pictures, like real trees in nature except that they’re drawn with the root at the top and the leaves at the bottom.

Terminology

```
node        -    a point in the tree. In these pictures, each node includes a label (value at each node)
root        -    the node at the top. Every tree has one root node
children    -    the nodes directly beneath it. Arity is the number of children that node has.
leaf        -    a node that has no children. (Arity of 0!)
```

Representation

We're going to use the Tree class from lecture.

Question 1:

The following exercises are about Trees. To use the Tree class,

In your terminal:

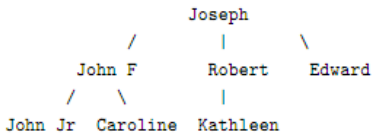
```
cp ~cs61a/public_html/sp12/labs/lab9/tree.py .
```

Open the python interpreter:

```
from tree import *
```

Take a moment to study the implementation of Trees. Look at the constructors, selectors, and printing methods. At the bottom, we give you some premade trees that you can work with for each section.

Now, after loading the file, the variable kennedy contains the following Tree:



Use the correct series of selectors to return the value “Caroline” from the tree.

Mutual Recursion (Your TA will take a moment to explain this)

Mutual Recursion occurs when you have a function that will deal with Trees, as well as a function that will deal with Forests (sequences of Trees). Both functions call each other to solve whatever problem is at hand.

```
def square_tree(tree):
    return Tree(square(tree.label), *square_forest(iter(tree)))

def square_forest(forest):
    new_forest = ()
    for tree in forest:
        new_forest += (square_tree(tree),)
    return new_forest
```

The variable t is defined in tree.py and is a Tree of integers that you can use to test your functions. Try out the following to see how it works!

```
>>> print(t)
_____________
>>> st = square_tree(t)
>>> print(st)
_____________
```

Exercise: Draw your interpretation of this tree on a piece of paper

Question 2:

Define the function tree_map which takes a function and a Tree as an argument and returns the equivalent Tree with the function applied to each node’s value. (Hint! This should be a similar but more general form of square tree)

```
>>> print(t)
_____________
>>> st = tree_map(square, t)
>>> print(st)
_____________
```

Question 3:

Define the function max_of_tree which takes in a Tree as an argument and returns the max of all of the values of each node in the Tree.

```
>>> print(t)
_____________
>>> max_of_tree(t)
_____________
```

Sets

A set is an unordered collection of distinct objects that supports membership testing, union, intersection, and adjunction.

Construction

```
>>> s = {3, 2, 1, 4, 4}
>>> s
{1, 2, 3, 4}
```

Adjunction

```
>>> s.add(5)
>>> s
{1, 2, 3, 4, 5}
```

Operations

```
>>> 3 in s
True
>>> 7 not in s
True
>>> len(s)
5
>>> s.union({1, 5})
{1, 2, 3, 4, 5}
>>> s.intersection({6, 5, 4, 3})
{3, 4}
```

For more detail on Sets you can go to the following link: PythonSets

Question 6:

Implement the union function for sets. Union takes in two sets, and returns a new set with elements from the first set, and all other elements that have not already have been seen in the second set.

```
>>> r = {0, 6, 6}
>>> s = {1, 2, 3, 4}
>>> t = union(s, {1, 6})
{1, 2, 3, 4, 6}
>>> union(r, t)
{0, 1, 2, 3, 4, 6}
```