CSE 341, Winter 2008, Assignment 6
Due: Thursday 6 March, 8:00AM

Last updated: February 26

Overview: You will define 3 Ruby classes for trees of strings as well as some additional methods.

1. Define 2 classes `Leaf` and `BinaryNode` with the following methods (sample solution is about 40 lines, many of which are just `end`):
   (a) `Leaf`'s `initialize` takes one argument (assumed to be a string, no need to check).
   (b) `BinaryNode`'s `initialize` takes two arguments, both assumed to be "trees of strings" (objects with the methods defined in this problem). These are the node's children.
   (c) `concatAll` takes no arguments and returns a single string that is all of a tree's strings concatenated together in left-to-right order.
   (d) In `BinaryNode`, define a *class method* (like a Java static method) `self.firstAlphabetical` that takes two strings and returns the string that comes first alphabetically. The `casecmp` method already in the `String` class makes this *very* easy.
   (e) `firstAlphabetical` takes no arguments and returns the string in the whole tree that comes first alphabetically.
   (f) `iterate` takes one argument of class `Proc` (i.e., something produced by `lambda {|x| ...}`) and calls its argument with each string in the tree.
   (g) In `BinaryNode`, define a *class method* `self.concatAll` that takes one argument, a "tree of strings", and returns all its strings concatenated together. Do *not* use the tree's `concatAll` instance method. Instead, use its `iterate` method. Hint: Use a local variable that starts with the empty string and gets imperatively updated to a longer string during the iteration.

2. Define a class `NaryNode` that is like `BinaryNode` except it can have any positive number of children. Sample solution is about 20 lines. Note:
   - `intialize` should take an array of trees. It should raise an error if the array's length is 0. Else it should store a *copy* of the array in a field. Each tree in the array is one of the node's children.
   - For `concatAll`, `firstAlphabetical` and `iterate`, use the `each` method of the `Array` class so your answers are at most a few lines long.

3. Now suppose you get tired of using `Leaf.new` all the time when building trees. Make it so that you can put strings in your trees directly rather than using the `Leaf` class at all. Do this by adding `concatAll` and `firstAlphabetical` methods to the built-in `String` class. Hint: The solution is perhaps a "trick" but *extremely* short — just think about what these methods should return. Do not bother adding an `iterate` method.

4. In a comment in your code, answer each of these questions in a few English sentences:
   (a) If you built a tree using just the `Leaf` and `BinaryNode` classes but you put integers at each leaf instead of strings, what would happen if you called the tree's `concatAll` method? Why?
   (b) If you used integers as in the previous problem but part of your tree was built with `NaryNode`, what would happen if you called the tree's `concatAll` method? Why?
   (c) Why does `NaryNode`'s `intialize` method make a copy of its argument? What could happen if it did not?
   (d) Why might adding methods to the `String` class be a poor design choice in a large application?

5. This problem has nothing to do with the trees defined above. Instead, you will write a Ruby version of the function `reachable_contacts` you wrote about in homework 2. Sample solution is 13 very dense lines (a few more lines is likely).

   - `reachable_contacts` should be defined outside of any explicit class.
   - `reachable_contacts` takes 2 arguments, a string `name` and a hash `all_people`. The hash should map names (i.e., strings) to arrays of names. (The provided test method `test_reach` defines one such hash.) If the hash maps string s to array a, we say that, "the friends of s are the names in a."
   - Your method should return an array of strings. For `reachable_contacts(n,h)` the array should hold `n`, the friends of `n`, the friends of friends of `n`, and so on, i.e., the names of all people "reachable via friend links" from `n`. The result must not have any repeated names. Order is unimportant.
   - Hints:
     - Follow the same basic algorithm as the code in homework 2.
     - Use a while loop rather than a recursive helper function and use two mutable local variables (an array that is the answer so far and an array for the "to-do collection" of names still to process).
     - Add elements to an array of answers with the `push` method of `Array`.
     - For the "to-do collection" of names, use an array of names (whereas the ML code used a `name list list`). On each iteration, use `pop` to remove one element from the array (this method returns `nil` if the array is empty). To add another array of names, use `+`.
     - To see if a string is already in an array, use the `any?` method of `Array`.

6. **Challenge Problem:** The versions of `concatAll` so far are inefficient in that for a tree with n strings, they create n different strings as intermediate results, each one longer than the previous one. Add different methods to produce the same result without creating all the intermediate strings. Hints: Make two passes through the tree. To make a string of length `len`, use `"x"*len`.

   Warning: The sample solution does *not* include a solution to the challenge problem.

Turn-in Instructions

- Put all your solutions in one file, `lastname_hw6.rb`, where `lastname` is replaced with your last name.
- The first line of your `.rb` file should be a Scheme comment with your name and the phrase `homework 6`.
- Go to `https://catalysttools.washington.edu/collectit/dropbox/djg7/1359` (link available from the course website), follow the "Homework 6" link, and upload your file.