Homework 9

March 25, 2014

Instruction

Write the answers to all questions in one text file of the name `hwk9.rb`. Make sure that your program can execute the test cases.

Ruby programming

This question is based on the previous homework of string trees.

Question 1

Define the `NaryNode` with the methods described below. A `NaryNode` has n children, where each child can be either a leaf, a binary node, or a nary node.

1. The class `NaryNode` has the same methods as those of `BinaryNode` except that `NaryNode` can have any positive number of children.

   (a) The `initialize` method of `NaryNode` should take an array of trees. It should raise an error if the size of the array is 0. It should store a copy of the array in a field. (Note that you can make a copy of an array using the `clone` method of the array class.) Each tree in the array is one of the node's children.

   (b) For `concatAll`, `firstAlphabetical`, and `iterate` methods, use `each` or `map` or `inject` method of the Array class so that your answers are at most a few lines long.

Question 2

Suppose you want to put strings in your trees directly rather than using the `Leaf` class. To accomplish this, add `concatAll`, `firstAlphabetical`, and `iterate` methods to the built-in `String` class. This is not really a good programming style but you should be aware that it is something Ruby is capable of.

Question 3

Define a Ruby module `TreeEnum` (a mixin) that contains the method `any?` and `inject`. These two methods assume the existence of an `iterate` method that takes a closure as argument and applies the closure to each leaf in the tree.

For example, you can invoke the method as `self.iterate(cls)`, where `cls` is a closure expression.

A module is defined as

```
module TreeEnum
  # define your methods here
end
```

Include this module in your `BinaryNode` and `NaryNode` class definitions so that we can call the `any?` and `inject` methods on objects of these classes.

To include the module in a class,

```
class MyClass
  include TreeEnum
end
```

1. The method `any?` takes a closure f as a parameter and f takes a parameter and returns either true or false. The method `any?` returns true if applying f to any of the leaf value results in true and it returns false otherwise.

   For example, let

   ```
   f = lambda {|x| x.start_with("great")}
   ```

   The expression `f.call("great day")` evaluates to `true`.

   If the tree `t1` contains the string `great`, then `t1.any?(f)` will return true.

2. The method `inject` takes two parameters: a closure f and an initial value `c`. (This method is similar to the `foldr` or `foldl` functions of ML).

   The closure f takes two parameter `acc` and `elm`. The parameter `acc` has `c` as the initial value and it stores the accumulative results of applying f to each value of the tree leaf. The value of tree leaf is passed to the parameter `elm`. The function `inject` will return the final result of applying f to the value of last tree leaf.

   For example, let

   ```
   f = lambda {|acc, elm| acc + elm.capitalize + " "}
   ```

   The expression `f.call("first", "word")` will return `"firstWord "`.

   If the tree `t1` contains the strings `"a"`, `"great"`, `"day"`, then the expression `t1.inject(f, "")` will return `"A Great Day "`.

Tests

Test program:

```
def test_print t2
  puts "t2.concatAll: " + t2.concatAll.to_s
  puts
  puts "t2.firstAlphabetical: " + t2.firstAlphabetical.to_s
  puts
  puts "t2.iterate(lambda { |s| puts s }):"
  t2.iterate(lambda { |s| puts s })
end

def test_tree
  l0 = Leaf.new "What "
  l1 = Leaf.new "a "
  l2 = Leaf.new "great "
  l3 = Leaf.new "day"
  l4 = Leaf.new "!"
  t0 = BinaryNode.new(l0,l1)
  t1 = BinaryNode.new(t0,l2)
  t2 = NaryNode.new([t1,l3,l4])

  test_print t2

  puts "\nThe following works after question 2\n\n"

  t2 = NaryNode.new([t1, "day", "!"])

  test_print t2

  puts "\nThe following works after question 3\n\n"

  puts "any word starting with 'great': " +
                                      t2.any?(lambda {|x| x.start_with?("great")}).to_s
  puts
  puts "capitalize: " + t2.inject((lambda {|acc, elm| acc + elm.capitalize + " "}), "")
end

test_tree
```

Result of the tests:

```
E:\doc\uwm\431\2014Fall\homeworks\9>ruby hw9.rb
t2.concatAll: What a great day!

t2.firstAlphabetical: !

t2.iterate(lambda { |s| puts s }):
What
a
great
day
!

The following works after question 2

t2.concatAll: What a great day!

t2.firstAlphabetical: !

t2.iterate(lambda { |s| puts s }):
What
a
great
day
!

The following works after question 3

any word starting with ‘great’: true

capitalize: What  A  Great  Day !
```