CSC148 Lab#8, winter 2016

learning goals

In this lab you will practice implementing recursive functions where the input is a **BinaryTree**. You may want to review lecture materials.

You should work on these on your own before Thursday, and you are encouraged to then go to your lab where you can get guidance and feedback from your TA . There will be a short quiz during the last 15 minutes of the lab, based on these exercises.

set-up

Open file binary_tree.py and save it under a new sub-directory called **lab08**. This file provides you with a declaration of class **BinaryTree**, headers and docstrings for the functions you will implement. Once you have read over the **_init_** method for class **BinaryTree**, you are ready to proceed to the implementation of the functions below. If you have questions, call your TA over.

implement parenthesize

This function takes an arithmetic expression tree and produces the equivalent parenthesized expression.

Read over the header and docstring for this function in **binary_tree.py**, but **don't** write any implementation until you fill in the steps below:

1. One of the examples in our docstring is simple enough not to require recursion. Write out an **if...** expression that checks for this case, and then returns the correct thing. Include an **else...** for when the tree is **not** so easy to deal with.

2. Suppose the call in the previous step gives you the correct answer according to the docstring: it returns a parenthesized string representing the arithmetic expression tree it is given. How will you combine the solutions for all the smaller instances to get a solution for **BinaryTree t** itself? Write code to return the correct thing.

Go over these three parts with your TA. After that, fill in the implementation in **binary_tree.py**, and see whether it works.

implement list_between

This function lists the nodes with values between the start and end values. Since it acts on a binary search tree, it does this efficiently, without traversing unnecessary nodes.

Read over the header and docstring for this function in **binary_tree.py**, but **don't** write any implementation until you fill in the steps below:

1. One of the examples in our docstring is simple enough not to require recursion. Write out an **if...** expression that checks for this case, and then returns the correct thing. Include an **else...** for when the tree is **less** easy to deal with.

2. Suppose the call in the previous step gives you the correct answer according to the docstring: it returns a list of nodes in binary search tree that are between the start and stop values. How will you combine the solutions to all the smaller instances to get a solution for **BinaryTree t** itself? Write code to return the correct thing. Put this code in the **else...** expression that you created in the first step.

Go over these three parts with your TA. After that, fill in the implementation in **binary_tree.py**, and see whether it works.

implement list_longest_path

This function returns a Python list with the values from a longest path in this tree.

Read over the header and docstring for this function in **binary_tree.py**, but **don't** write any implementation until you fill in the same steps as you did for the last two functions. At the end, review the three steps with your TA before filling in the implementation.

additional exercises

Good additional exercises are to try implementing any function or method from a general **Tree** on a **BinaryTree**. As usual, the last 15 minutes of the lab will consist of a 15-minute quiz.