CSC148 Lab#8, winter 2017

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 25 minutes of the lab, based on these exercises.

set-up

Open file ex7.py and save it under a new sub-directory called **lab8**. This file provides you with a declaration of class **BinaryTree**, headers and docstrings for the functions you will implement. As well pylint.txt is a configuration file for **python_ta**.

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 **ex7.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 **ex7.py**, and see whether it works.

implement list_between

This function lists the nodes with values between the start and end values in a binary search tree.

Read over the header and docstring for this function in **ex7.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 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 **ex7.py**, and see whether it works. **Be sure** your solution uses the fact that this is a **binary search tree** to avoid looking at subtrees where values are smaller than **start** or larger than **stop**

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 **ex7.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.

implement count_shallower

This function returns the number of nodes in nodes with depth less than n. That means that the recursive function needs to carry around a parameter that keeps track of its depth: n itself. Notice that nodes that have depth shallower than n with respect to the root have depth $n - 1$ with respect to the root's children.

Read over the header and docstring for this function in **ex7.py**, but **don't** write any implementation until you fill in the same steps as you did for other functions. At the end, review the three steps with your TA before filling in the implementation.

additional exercises

Submit completed **ex7.py** on MarkUs by 10 p.m. March 12. You can find some unit tests, ex7_test.py, that test the same things as the doctest examples.

Good practice additional exercises are to try implementing any function or method from a general **Tree** on a **BinaryTree**, using **left** and **right** instead of **children**. As usual, the about 25 minutes before the end of the lab your TA will have you write a quiz, and during the last 10 minutes you will discuss your answers before the quiz is collected.