CSC148 Lab#7, winter 2018

learning goals

In this lab you will practice implementing recursive functions where the input is a general **Tree**, as presented in lecture. 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 an online quiz during the second hour of lab, based on these exercises.

set-up

Open file ex6.py and save it under a new sub-directory called **lab6**. This file provides you with a declaration of class **Tree**, headers and docstrings for the functions you will implement, as well as two utility functions we think you'll find useful:

gather_lists: There will be cases where you have a list whose elements are sub-lists, and what you'd really like is to concatenate them into a single list. That's what **gather_lists** does.

descendants_from_ list: This function makes it less laborious to build a **Tree** from a list of data. It simply fills in the tree, level by level, from a list, until it runs out of elements. You'll need to save csc148_queue.py in **lab6** to make this work.

Once you have familiarized yourself with the **__init__** method for class **Tree**, you are ready to proceed to the implementation of the functions below.

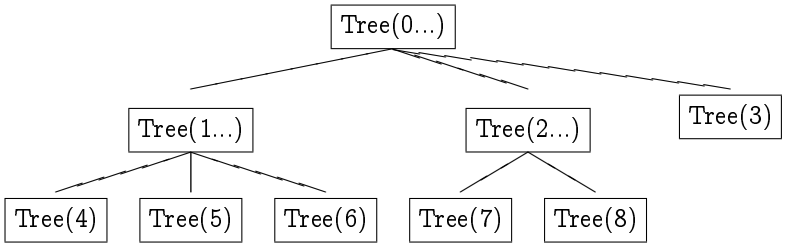
implement list_internal

This function will list all the **values** from internal nodes of **Tree t**. The order of the list is not specified, and there is no need to remove duplicates.

Read over the header and docstring for this function in **ex6.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. Below is a picture of a larger **Tree**, with several levels. Consider a function call **list_internal(t)**, assuming that **t** is a reference to the **Tree**. Are there one or more smaller trees for which it would be helpful to know the list of values that they contain? Which smaller trees are they? Write an example of a function call **list_internal** on one of these smaller trees. You can access these trees through the variable **t**.



3. Suppose the call in the previous step gives you the correct answer according to the docstring: it returns a list of the values in the tree it is given. How will you combine the solutions for all the smaller instances to get a solution for **Tree t** itself? Write code to return the correct thing. **Hint:** You may want **gather_lists** here. Put this code in the **else...** expression that you created in the first step.

Go over these three parts. After that, fill in the implementation in **ex6.py**, and see whether it works. A good approach is to comment-out all the functions you have not yet implemented, so that you will see only doctest results for the one you're working on.

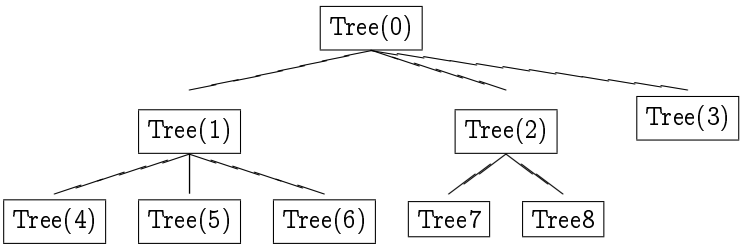
arity

This function returns the maximum branching factor of this tree.

Read over the header and docstring for this function in **ex6.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. Below is a picture of a larger **Tree t**, with several levels. Consider a function call that passes **t** as an argument. Are there one or more smaller trees for which it would be helpful to know the arity (branching factor) of? Which smaller trees are they? Write an example of a function call on one of these smaller trees. You can access these trees through the variable **t**.



3. Suppose the call in the previous step gives you the correct answer according to the docstring: it returns the arity of the tree it is given. How will you combine the solutions to all the smaller instances to get a solution for **Tree 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. After that, fill in the implementation in **ex6.py**, and see whether it works.

contains_test_passer

This function returns whether at least one of the values in this tree returns True for the boolean function the user passes in.

Read over the header and docstring for this function in **ex6.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 before filling in the implementation.

additional exercises

There are a couple of additional exercises in ex6.py as well as **arity**,**list_internal** and **contains_test_passer**. Implement all functions **recursively**. These are likely to be very similar to the material you are quizzed on during the second hour of lab. We've added test_count_internal_basic.py, test_count_leaves_basic.py, test_sum_internal_basic.py, test_sum_leaves_basic.py, to help increase your confidence that your code works.