CSC148 Summer 2018: Midterm 2 Practice

The questions in this practice encompass more than would be on a midterm. The questions are designed to cover a breadth of topics. Within each category are multiple questions that address various topics, and then 1 or 2 questions at the end that is more "test" styled (i.e. a question that would be fair for a test) at the end.

In addition to answering the questions within this document, you should work through the labs, past exams, and the extra recursion practice problems.

Recursion on 'Nested Lists'

For this section, "nested list" refers to a list that may contain additional lists as an element. Saying "nested list of ints" means that the elements of the list are either ints and/or more nested lists of ints.

1. In general terms, what is a base case?
   The simplest case possible where we don't need to use recursion.

2. Suppose we have a function called get_sum() which takes in an int or a nested list of ints and returns the sum of all ints in it. What would the base case for this be?
   When the item passed in is an int, there's nothing for us to make recursive calls on/that we need to simplify further.

3. When writing recursive solutions, we assume our recursive calls works perfectly. For the question from (2), what would a list look like for which all recursive calls would reach the base case?
   [1, 2, 3, 4]
   All of the items in this list, when we call get_sum on them, would reach the base case.

4. Continuing from (3), what would we expect to do with the results of the recursive calls?
   Add them all together! :)

5. Continuing from (4), create a list that contains the list from (2) as well as elements on which recursive calls would reach the base case immediately. What would we expect back from those recursive calls? If we followed the steps from (4), would we still get the answer we want in the end?
   [[1, 2, 3, 4], 5, 6]
   From the recursive call to get_sum([1, 2, 3, 4]), we would expect 10 back.
   get_sum(5) should return 5
   get_sum(6) should return 6
   Adding all of those together, we would get 21.

6. When doing recursion, we always break our problem into smaller and easier to solve subproblems, which eventually reach a base case. For the question from (2), what would these smaller problems be? How are they simpler than the original problem?
   The smaller problems are each of the elements -- our original problem would be the entire list itself.
   E.g. Suppose we had the list:
   [[1, 2], [[3]], 4, [5, [6]]]

   Dealing with each of these:
   [1, 2] [[3]] 4 [5, [6]]
   Is a lot easier than dealing with the original list!

7. Suppose we have a function called count_elements_in which takes in an int or a nested list of ints, as well as a list of ints, and returns a list of all of the elements that appear in the list of ints passed in.

   For example, suppose we call count_elements_in(5, [1, 2, 3]): We would expect 0 to be returned because 5 is not in [1, 2, 3].

   If we called count_elements_in([1, 2, [3, 4], [5, [3], 2], [[1]]], [2, 3]) we would expect 4 to be returned since there are 4 elements that appear in the list [2, 3].

   a) What would we expect to get back from calling count_elements_in([[1, 2], [3, [[4], 1], [5], [[7, 8], 1], 5], 3], [1, 3, 5])?
      7
   b) What recursive calls would we make on this list?
      count_elements_in([1, 2], [1, 3, 5])
      count_elements_in([3, [[4], 1], [5], [[7, 8], 1], 5], [1, 3, 5])
      count_elements_in(3, [1, 3, 5])
   c) What would we expect back from those recursive calls?
      count_elements_in([1, 2], [1, 3, 5]) should return 1
      count_elements_in([3, [[4], 1], [5], [[7, 8], 1], 5], [1, 3, 5]) should return 5
      count_elements_in(3, [1, 3, 5]) should return 1

8. Implement count_elements_in() from (7).
   See midterm_practice_part_1_solutions.py

9. Suppose we have a function called sum_at_depth which takes in an int or a nested list of ints, as well as an int representing a depth, and returns the sum of all ints at that depth. 'depth' refers to how many lists an element is nested inside -- i.e. an int has a depth of 0, but an int like [5] has a depth of 1 since it's nested inside 1 list.

   Suppose we have the following nested list:
   [1, [3, [4, 5, [[6]]], 7], [2, [10]]]

   And suppose we want the sum of the items a depth of 2.

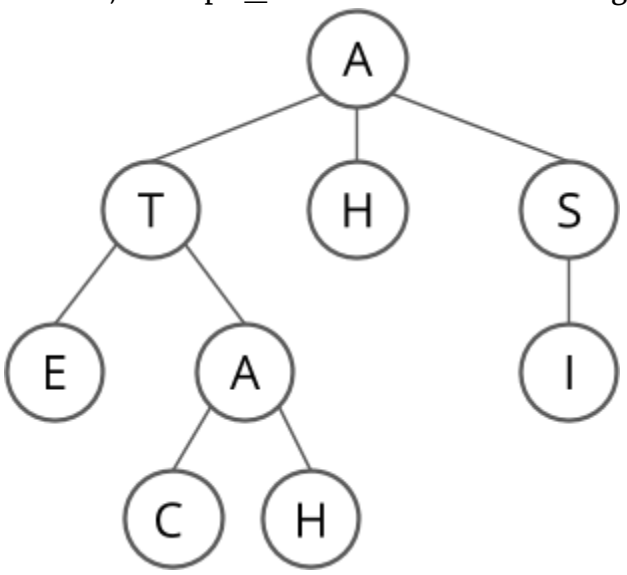
   a) What would we expect back from calling sum_at_depth([1, [3, [4, 5, [[6]]], 7], [2, [10]]], 2)?
      12 (3 + 7 + 2)
   b) What recursive calls would we make on this list?
      sum_at_depth(1, 1)
      sum_at_depth([3, [4, 5, [[6]]], 7], 1)
      sum_at_depth([2, [10]], 1)
   c) What would we expect back from those recursive calls?
      sum_at_depth(1, 1) should return 0
      sum_at_depth([3, [4, 5, [[6]]], 7], 1) should return 10
      sum_at_depth([2, [10]], 1) should return 2

10. Implement sum_at_depth() from (9).
    See midterm_practice_part_1_solutions.py

Trees

The file midterm_tree.py contains the Tree that you'll be provided during the midterm. This is simply the Tree from lecture with only the ___init___ provided.

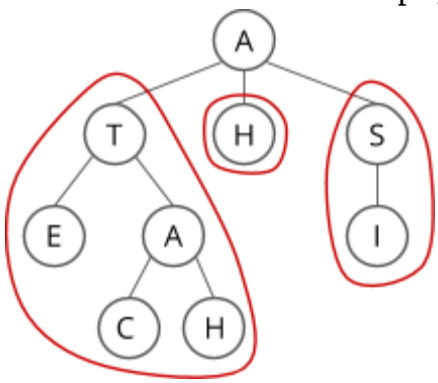
For the questions in this section, 'example_tree' refers to the following Tree:



1. What is the height of example_tree?
   4

2. a) What is the depth of the node with value 'C' in example_tree?
      3
   b) What is the depth of the node with value 'S' in example_tree?
      1
   c) What is the relationship between max depth and height?
      Max depth = height - 1
      (Or height = max depth + 1)

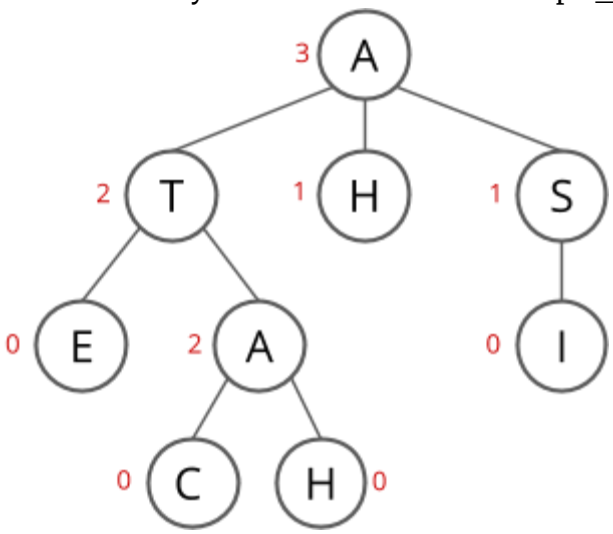
3. What are the subtrees of example_tree?



   Those 3 subtrees.

4. a) What is the arity of example_tree?
      3

b) What is the arity of each of example_tree's children?
2, 0, and 1.
c) What is the arity of all of the nodes in example_tree?



5. If we were to do a pre-order traversal of example_tree, in what order would values be processed?
A, T, E, A, C, H, H, S, I

6. If we were to do a post-order traversal of example_tree, in what order would values be processed?
E, C, H, A, T, H, I, S, A

7. If we were to do a level-order traversal of example_tree, in what order would values be processed?
A, T, H, S, E, A, I, C, H

8. Write the code needed to create example_tree in Python.
See midterm_practice_part_1_solutions.py

9. 2 Trees are equal if they have the same values and all of their subtrees are equal. Implement an ___eq___ method for a Tree.
See midterm_practice_part_1_solutions.py

10. Suppose we want to write a method of a Tree called able_to_spell() which takes in an string and returns whether there's a chain of nodes that can spell that string from lowest to highest node (i.e. in the order some node -> its parent -> its parent). For example, calling example_tree.able_to_spell('IS') should return True since there's a node with the value 'I' and its parent which has an 'S'
I messed up the wording of this question. I should have said
"If there's node in a path such that we could spell that string from lowest to highest."
I.e. "HT" would be spellable for example_tree -- I tried to simplify the wording a bit much though.

If you do want a solution to the one I actually ended up stating, that would require a helper function that gets all paths from a leaf to the root, and checks whether the word appears in any of those paths.
(Or alternatively, a helper function that takes in an additional parameter to decide

whether to start searching in a consecutive path or not.)

Suppose we're calling example_tree.able_to_spell('CAT')

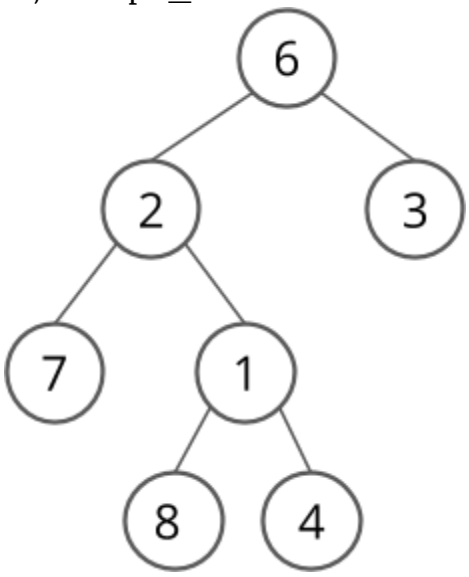
a) What are the subtrees of example_tree?
   See the answer to (3)
b) What are the recursive calls we would make?
   (subtree rooted at 'T').able_to_spell('CAT')
   (subtree rooted at 'H').able_to_spell('CAT')
   (subtree rooted at 'S').able_to_spell('CAT')
c) What do we expect back from the recursive calls?
   (subtree rooted at 'T').able_to_spell('CAT') should return True
   (subtree rooted at 'H').able_to_spell('CAT') should return False
   (subtree rooted at 'S').able_to_spell('CAT') should return False

11. Implement the able_to_spell() method from (10).
    See midterm_practice_part_1_solutions.py

Binary Trees

The file midterm_binary_tree.py contains the BinaryTree that you'll be provided during the midterm. This is simply the BinaryTree from lecture with only the ___init___ provided.

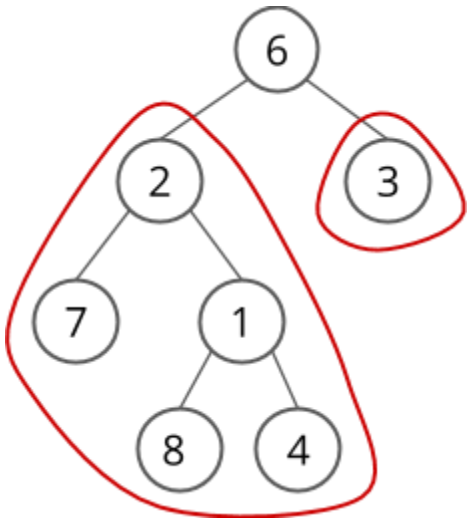
For the questions in this section, 'example_tree' refers to the following BinaryTree:



1. What is the height of example_tree?
   4

2. d) What is the depth of the node with value 4 in example_tree? 3
   e) What is the depth of the node with value 7 in example_tree? 2
   f) What is the relationship between max depth and height?
      Same as the answer for Tree. :)

3. What are the subtrees of example_tree?



4. What is the difference between a Tree and a BinaryTree?
   Tree: Has a **list** of subtrees, can have any number of subtrees.
   BinaryTree: Has a left and a right subtree (possibly only one or neither). Must have <= 2 subtrees.

5. If we were to do a pre-order traversal of example_tree, in what order would values be processed?
   6, 2, 7, 1, 8, 4, 3

6. If we were to do a post-order traversal of example_tree, in what order would values be processed?
   7, 8, 4, 1, 2, 3, 6

7. If we were to do a level-order traversal of example_tree, in what order would values be processed?
   6, 2, 3, 7, 1, 8, 4

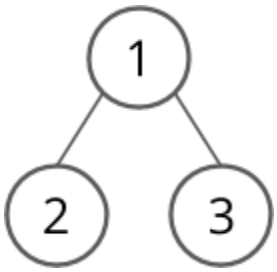
8. If we were to do an in-order traversal of example_tree, in what order would values be processed?
   7, 2, 8, 1, 4, 6, 3

9. Write the code needed to create example_tree in Python.
   See midterm_practice_part_1_solutions.py

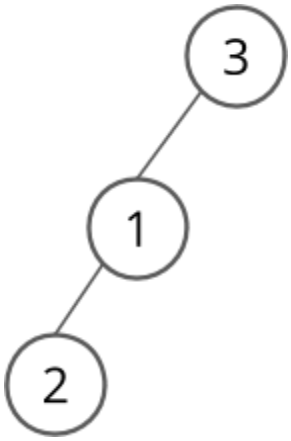
10. 2 BinaryTrees are equal if they have the same values and their left and right subtrees are equal. Implement an ___eq___ **method** for a BinaryTree.
    See midterm_practice_part_1_solutions.py

    (This will be the only method we implement for BinaryTrees; but you could write a function instead.)

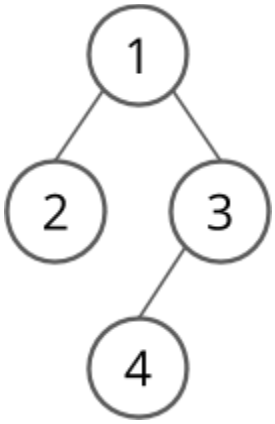
11. For this question, the term 'rotate' means to rotate a BinaryTree such that the new root is the root of one of the children. For example, suppose we have the following BinaryTree:



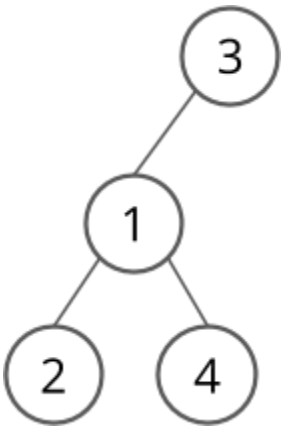
Rotating it left means to rotate it counter-clockwise, so that the right child is the new root, and the original root is its new left-child:



If we had a left subtree already, i.e.:



Then this subtree becomes the right subtree of what used to be the root (since it lost its right child):



Write a function called rotate_left() which takes in a BinaryTree and left-rotates it, returning the new root.
See midterm_practice_part_1_solutions.py