CSC148, Exercise #3
due 9:59 p.m. March 13th, 2014

There are two parts to this exercise. For each part, there is a single file you must submit — make sure that you read the submission instructions carefully!

general advice

Before you begin coding, make sure that you understand everything in this handout. Remember that the point of these exercises is not to get a "working" program by any means necessary — it is to ensure that you understand how to write the desired code. If you succeed in writing something that works but you do not understand why, then you have failed, no matter what grade you receive!

In particular, if there is something you do not quite understand, please, for your own sake, take the time to read the course notes and readings, do some searching on Google and ask on the course forum or during office hours to make sure you find out. You will find some related code from Week #6 of the course.

background

Both parts of this exercise concern binary trees. All the keys (that is, the data stored in the tree nodes) are expected to be integers. Even though the functions in both parts should work on some kinds of non-integer data, we will use only integers in testing.

You have to write two functions — one each in Part A and Part B. Both functions should assume that a tree is a Python structure of the following form:

TreeList: is either `None` or a 3-element list of the following form:

- Element 0 is an integer
- Element 1 is a `TreeList`
- Element 2 is a `TreeList`

Here are some examples of `TreeLists`:

```
None
```

```
[2, None, None]
```

```
[3, [7, None, [2, None, None]], [5, None, None]]
```

part A

Consider re-constructing a binary tree from its pre-order and in-order traversals. For example, if the lists of keys resulting from the traversals are [10, 6, 8, 12, 11, 15] (pre-order) and [8, 6, 12, 10, 11, 15] (in-order), then we can see that:

- The key at the root is 10, because that is how the pre-order traversal begins.
- There are three nodes in the left sub-tree, because there are three keys listed before 10 in the in-order traversal.
- Many other similar inferences can be made...

Submit a file called `e3a.py` that defines a function `make_tree(preorder, inorder)`. This function must return a `TreeList` containing the binary tree whose pre-order and in-order traversal lists are given as arguments. If either preorder or inorder is empty, then you can assume that both arguments are empty and you should return `None`.

We promise not to use test cases containing two keys with the same value. Also, your function can assume that the parameters represent correct pre-order and in-order traversals of the same binary tree. We will not test your code with invalid parameters.

Our test programs will access your function by including the line:

```
from e3a import make_tree
```

Hint: Think before you code! In particular, work through a number of small (and large) examples by hand to figure out exactly how to solve the problem, before you attempt to write the function.

part B

Submit a file called `e3b.py` that defines a function `sum_to_deepest(t)` that returns the sum of the keys from the root to the deepest leaf in `TreeList t`. If there are two deepest leaves, return the larger sum; if `t` is `None`, return `0`. In the tree used as an example in Part A, the sum would be:

$$10 + 11 + 15 = 36$$

In this problem, you may find it helps to define a helper function that returns both the depth of the deepest leaf and the sum along the path to that leaf.

Our test programs will access your function by including the line:

```
from e3b import sum_to_deepest
```

Hint: Just like for Part A, work through a number of small (and large) examples by hand to figure out exactly how to solve the problem, before you attempt to write the function.