CS 61A Exam-Prep Section 3

Trees, non-mutative lists, tree recursion, environment diagrams

Fall 2015, Midterm 2, #3a — tree recursion, trees

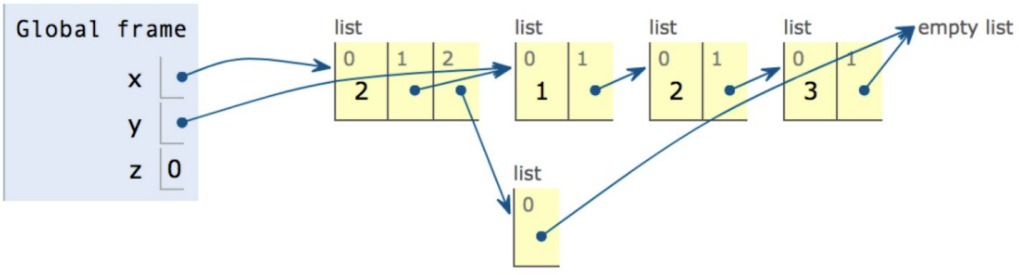
3. (24 points) Return of the Digits

(a) (4 pt) Implement `complete`, which takes a `Tree` instance `t` and two positive integers `d` and `k`. It returns whether `t` is *d-k-complete*. A tree is *d-k-complete* if every node at a depth less than `d` has exactly `k` branches and every node at depth `d` is a leaf. *Notes:* The depth of a node is the number of steps from the root; the root node has depth 0. The built-in `all` function takes a sequence and returns whether all elements are true values: `all([1, 2])` is `True` but `all([0, 1])` is `False`. `Tree` appears on the Midterm 2 Study Guide.

```
def complete(t, d, k):
    """Return whether t is d-k-complete.

    >>> complete(Tree(1), 0, 5)
    True
    >>> u = Tree(1, [Tree(1), Tree(1), Tree(1)])
    >>> [ complete(u, 1, 3)  ,  complete(u, 1, 2)  ,  complete(u, 2, 3) ]
    [True, False, False]
    >>> complete(Tree(1, [u, u, u]), 2, 3)
    True
    """
    if not t.branches:

        return ____________________________________________________________

    bs = [_____________________________________________________________]

    return ____________________________________________________ and all(bs)
```

Spring 2018, Exam-Prep 03, #1 — environment diagrams, non-mutative lists

1. Translating a List Diagram to Code



Fill in the following blanks so that after all lines have been executed, the environment looks as in the diagram above. You may not use numerals or mathematical operators in your solution.

```
x, y, z = 1, 2, 3
y = ____________________________________________________
x = ____________________________________________________
z = ____________________________________________________
```

Spring 2015, Midterm 2, #3c — tree recursion, trees

(c) (4 pt) Implement `closest`, which takes a `Tree` of numbers `t` and returns the smallest absolute difference anywhere in the tree between an entry and the sum of the entries of its branches. The `Tree` class appears on the midterm 2 study guide. The built-in `min` function takes a sequence and returns its minimum value. *Reminder*: A branch of a branch of a tree `t` is *not* considered to be a branch of `t`.

```
def closest(t):
    """Return the smallest difference between an entry and the sum of the
    entries of its branches.

    >>> t = Tree(8, [Tree(4), Tree(3)])
    >>> closest(t) # |8 - (4 + 3)| = 1
    1
    >>> closest(Tree(5, [t])) # Same minimum as t
    1
    >>> closest(Tree(10, [Tree(2), t])) # |10 - (2 + 8)| = 0
    0
    >>> closest(Tree(3)) # |3 - 0| = 3
    3
    >>> closest(Tree(8, [Tree(3, [Tree(1, [Tree(5)])])])) # |3 - 1| = 2
    2
    >>> sum([])
    0
    """
    diff = abs(_____________________________________________________________)


    return min(_____________________________________________________________)
```

Custom Question — tree recursion, trees, non-mutative lists

```
def is_path(t, path):
      """Return whether a given path exists in a tree, beginning
            at the root.

      >>> t = tree(1, [
             tree(2, [tree(4), tree(5)]),
             tree(3, [tree(6), tree(7)])
          ])
      >>> is_path(t, [1, 2])
      True
      >>> is_path(t, [1, 2, 4])
      True
      >>> is_path(t, [2, 4])
      False
      """
      if _________________________________:
            return False
      if _________________________________:
            return True
      return
any([_____________________________________________________________])
```

(b) (4 pt) Fill in the environment diagram that results from executing the code below after the entire program is finished. No errors occur during the execution of this example.

A complete answer will:

- Add all missing values created or referenced during execution.
- Show the return value for each local frame.

```
1 def scramble(egg):
2     return [egg, over(egg)]
3
4 def over(easy):
5     easy[1] = [[easy], 2]
6     return list(easy[1])
7
8 egg = scramble([12, 24])
```