Exam Prep Section 7 - CS61A Spring 2018

Worksheet 7: Object-Oriented Programming Trees & Linked Lists

Problem 1

```
def linky_paths(t):
     “””Takes in a tree, t, and modifies each label to be the path from that node
     to the root
     >>> t = Tree(1, [Tree(2)])
     >>> linky_paths(t)
     >>> t
     Tree(Link(1), [Tree(Link(2, Link(1))]”””
     def helper(t, path_so_far):
            t.label = Link(t.label, path_so_far)
           for b in t.branches:
                 helper(b, t.label)
     helper(t, Link.empty)
```

Problem 2

```
def find_file_path(t, file_str):
    """Returns the full path of a file that we search for if the file exists. If the
     file does not exist, then return None"""
     >>> t = Tree('data', [Tree('comm', [Tree('dummy.py')]), Tree('ecc',
     [Tree('hello.py'), Tree('file.py')]), Tree('file2.py')])
     >>> find_file_path(t, 'file2.py')
     '/data/file2.py'
     >>> find_file_path(t, 'dummy.py')
     '/data/comm/dummy.py'
     >>> find_file_path(t, 'hello.py')
     '/data/ecc/hello.py'
     >>> find_file_path(t, 'file.py')
     '/data/ecc/file.py'
    def helper(t, file_str, path_so_far):
        if t.is_leaf() and t.label == file_str:
            return path_so_far + '/' + t.label
        elif t.is_leaf():
            return
        for b in t.branches:
            result = helper(b, file_str, path_so_far + '/' + t.label)
            if result:
                return result
    return helper(t, file_str, '')
```

Problem 1

```
def convert_to_string(link):
       “””Converts a linked list that contains strings to a file path”””
       >>> link = Link(‘data’, Link(‘file2.py’))
       >>> convert_to_string(link)
       ‘/data/file2.py
       if link is Link.empty:
         return ''
       return '/' + link.first + convert_to_string(link.rest)
```

Problem 2

```
def all_paths_linked(t):
       “““Returns a list of all paths from root to leaf in a tree with one
       catch -- represent each path as a linked list
       >>> t1 = Tree(1, [Tree(2), Tree(3)])
       >>> t2 = Tree(1, [Tree(2), Tree(3, [Tree(4), Tree(5)])])
       >>> all_paths(t1)
       [Link(1, Link(2)), Link(1, Link(3))]
       >>> all_paths(t2)
       [Link(1, Link(2)), Link(1, Link(3, Link(4))), Link(1, Link(3, Link(5)))]”””
       if t.is_leaf():
              return [Link(t.label)]
       result = []
       for branch in t.branches:
              result = result + [Link(t.label, path) for path in
                          all_paths_linked(branch)]
       return result
```

Problem 3

```
def find_file_path2(t, file_str):
    """Returns the full path of a file that we search for if the file exists. If the
      file does not exist, then return None
      For this question, use the definition of all_paths_linked and
      convert_to_string"""
      >>> t = Tree('data', [Tree('comm', [Tree('dummy.py')]), Tree('ecc',
      [Tree('hello.py'), Tree('file.py')]), Tree('file2.py')])
      >>> find_file_path2(t, 'file2.py')
      '/data/file2.py'
      >>> find_file_path2(t, 'dummy.py')
      '/data/comm/dummy.py'
      >>> find_file_path2(t, 'hello.py')
      '/data/ecc/hello.py'
      >>> find_file_path2(t, 'file.py')
      '/data/ecc/file.py'

      for link in all_paths_linked(t):
              original = link
              while not link is Link.empty:
                  if link.rest is Link.empty and link.first == file_str:
                      return convert_to_string(original)
                  link = link.rest
```

Problem 4

```
def skip(lnk, n):
    '''Given a linked list LNK and a number N where N > 1, mutate LNK
    such that every Nth element is skipped.
    >>> lnk = Link(1, Link(2, Link(3, Link(4, Link(5, Link(6))))))
    >>> skip(lnk, 2)
    >>> lnk
    Link(1, Link(3, Link(5)))
    >>> lnk2 = Link(1, Link(2, Link(3, Link(4, Link(5, Link(6))))))
    >>> skip(lnk2, 4)
    >>> lnk2
    Link(1, Link(2, Link(3, Link(5, Link(6)))))
    '''
    count = 1
    def skipper(lst):
        nonlocal count
        count += 1
        if lst is Link.empty or lst.rest is Link.empty:
            return
        elif count % n == 0:
            lst.rest = lst.rest.rest
            count = 1
        skipper(lst.rest)
    skipper(lnk)
```