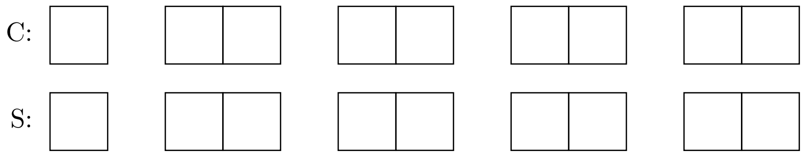
CS 61A Structure and Interpretation of Computer Programs
Fall 2016 Midterm 2

Post-MT1 topics that you'll want to have down cold (not necessarily a comprehensive list, but you better know all about these!): nonlocality, OOP, data abstraction, trees, linked lists, order of growth, mutation

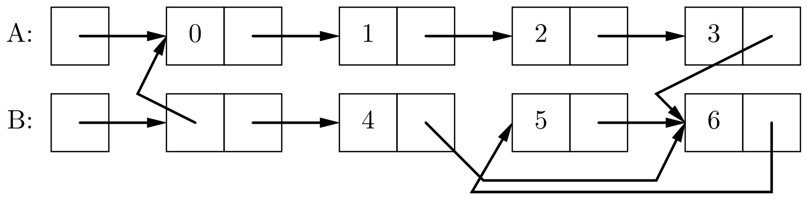
1. (4 points) Linked List Basics

For each of the following code fragments, add arrows and values to the object skeletons to the right to show the final state of the program. Single boxes are variables that contain pointers. Double boxes are `Links`. Not all boxes will necessarily be used. (Note: I suggest doing it on scratch paper before filling in the boxes.)

```
C = Link(1, Link(6))
C.rest.rest = Link(C.first, \
        Link(C.rest.first))
S = C.rest.rest
C.rest.rest = C.rest.rest.rest
C.rest.rest.first = C.rest
C, C.rest.rest = S.rest.first, C
```



Fill in the code to create the linked lists below. No multiple assignment allowed. Incidentally, as a general rule (i.e. in real life) try not to set `first` elements to other linked lists.



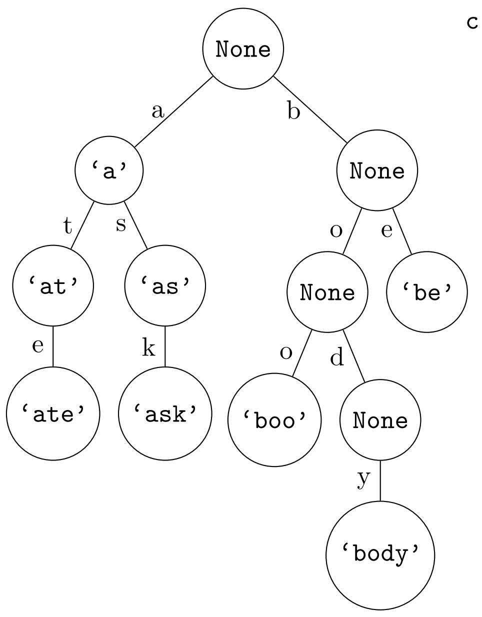
```
A = Link(0, Link(1, Link(2, Link(3))))

B = __________________________________________________________________

B.______________________________ = _____________________________________

B.______________________________ = _____________________________________
```

2. (6 points) There Is No Trie

Say we had a tree that stored English words character-by-character. The first level of the tree would contain the first character of a word, the second level the second character... and so on so forth. At the end of a complete word in the tree, we would store the full word (using all the characters up to that point) as the node value. If the string of characters leading up to some node was NOT a word, we would store `None` as its value.

Let's make this tree a reality. To facilitate such an undertaking, we edit our `Tree` class so that branches are stored as a dictionary from characters to `Trees`.

We include on the next page an example of a tree containing the strings {`body, be, boo, ate, ask, at, as, a`}, along with the modified `Tree` definition. Here, edges are marked with the key in the `branches` dictionary that they correspond to.



```
class Tree:
    def __init__(self, root=None, branches={}):
        self.root = root
        self.branches = branches

    def insert(self, chars, word):
        if len(chars) == 0:
            self.root = word
            return
        if chars[0] not in self.branches:
            self.branches[chars[0]] = Tree(None, {})
        self.branches[chars[0]].insert(chars[1:], \
                word)

    def autocomplete(self, prefix):
        ...

    def get_all_words(self):
        ...
```

To create the tree on the left, for instance, you could execute `tree = Tree(); [tree.insert(list(word), word) for word in ['ate', 'ask', 'boo', 'body', 'be', 'a', 'at', 'as']]`. (Obligatory note: you normally shouldn't use list comprehensions for stuff like this. I just didn't want the quiz to be three pages.)

We can process our word tree in many interesting – and efficient – ways. One example is autocompletion. *Fill in the blanks below so that* `autocomplete` *(a method of our* `Tree` *class!) returns a list of full words pertaining to the given prefix. For example,* `tree.autocomplete('bo')` *would return* `['boo', 'body']` *if* `tree` *were the tree from above. You will probably need to implement and use the* `get_all_words` *method, which returns a list of all of the words in a tree.*

```
def autocomplete(self, prefix):
    """Returns a list of full-word completions for the given prefix."""
    if len(prefix) == 0:

        __________________________________________

    if prefix[0] not in self.branches:

        __________________________________________

    ______________________________________________

def get_all_words(self):
    """Returns all words contained within the current tree."""
    words = []

    ______________________________________________

        words.append(self.root)

    for branch in self.branches.values():

        __________________________________________

    return _______________________________________
```