Question 1: Running Code from Lecture Notes

On the course website, I collected all the code from the lecture notes on binary search trees. I changed InOrderWalk so that it collects its result in a string, and returns the string.

Write some test code to show that Insert, Delete, Search, Min, Succ, and InorderWalk work. You can use a constant for the tree (and the value of keys), but do not use a variable or constant in creating the nodes. In other words, when you call Node to create a new node, insert it immediately into the tree, without saving it to a variable.

Make sure that you test your code on a tree of a non-trivial size. Make sure that InOrderWalk, Min, Search work both from the root node and some other node.

Do not turn anything in for this question.

Question 2: Two Classes

The code used two classes, one for Tree and one for Node. InOrderWalk is a method of Node. What are the advantages of having it a method of Node, rather than Tree?

Can InOrderWalk be used to produce a walk of an entire tree? If so, how?

Question 3: str Method

Create a magic method for str. This will work similar to InOrderWalk, except that you should show the hierarchy of the tree using brackets. Consider the following tree.

```
   6
  /  \
 3   9
 /  /  \
2 7 11
```

When called on the subtree of 2, it should return (2). When called on the subtree of 3, it should return ((2)3). When called on the subtree rooted at 6, it should return (((2)3)6((7)9(11))).

Hand in a copy of your str code.

Question 4: Insert

Here is the code from hw2.py for the insert method.

```python
def Insert(self, z):
    y = None
    x = self.root
    while x is not None:
        y = x
        if z.key < x.key:
            x = x.left
        else:
            x = x.right
    return y
```
This code obscures what happens when we insert into an empty tree. Change the code for Insert so that it deals with the empty tree case first, and then returns. Make sure you do not leave any of the existing code that deals with the empty case in the rest of the code.

Turn in your code.

**Question 5: Insert Order**

Say you have the numbers 1 through 7.

What order must you insert them to get a tree with all elements strictly on the rightmost branch?

What order must you insert them to get a tree with all elements strictly on the leftmost branch?

What order must you insert them to get a strictly balanced tree?

Hand in a written explanation.

**Question 6: Height of Tree**

Create a node method called Height() that determines the length of the longest path from the node to a leaf node. We will refer to this as the height of the node. To get the height of the tree, simply pass to it the root of the tree. The height of a leaf node (no right or left child) will be 1.

Hand in a copy of your code. If a tree has \( n \) nodes in it, and has a height of \( h \), how long will the code take? Is this a \( \Theta \), \( O \), or \( \Omega \)? You do not have to prove your answer.

**Question 7: Minimum Height of a Tree**

If you have a tree of size 1023 (\( 2^{10} - 1 \)), what is the smallest height it can have?

**Question 8: Randomly built trees**

Create a function called BuildTree1023. The function should build a tree from the numbers 1 to 1023, such that each number just occurs once, and the order that the numbers are inserted is random. It should take no arguments, and it should return the tree (actually a pointer to the tree object). You should import the random package.

Create a function called BuildTrees that computes the average height of trees that BuildTree1023 makes. It should do this by calling BuildTree1023 500 times and determining the height of each tree that is built. Have it also compute the minimum and the maximum heights seen. The function should take no inputs and it should print the minimum, average, and maximum heights.

Turn in the code for both functions.
**Question 9: Performance**

Run BuildTrees from the previous question and report on the average height of the trees built. Compare the average with the minimum possible height that you computed earlier and with the worse case, where the tree has no branching.

**Prelude to String Alignment**

**Question 10: All Paths**

Create a node method called PrintAllPaths that will print all paths in the binary tree from the node to all leaf nodes (nodes with no children), where each path is a list of the node names.

Consider the following tree:

```
    6
   / \
  3   9
 / \  / \  
2 5 7 11  
 /     
4 12
```

PrintAllPaths called on node 6 should print the following.

```
[6,3,2]
[6,3,5,4]
[6,9,7]
[6,9,11,12]
```

PrintAllPaths will be similar to InOrderWalk. PrintAllPaths should take two arguments. Since this is a method for the Node class, the first argument will be self. The second argument will be ancestors that are higher up in the tree. When you call PrintAllPaths, you should pass in an empty list. (Or you can have the second argument have a default value.)

Hand in your code.

**Web Resource**

A number of the homeworks will have questions under the title *Web Resource*. These questions were inspired by an email that I received from a PhD student from another department that was trying to scrape information from a web resource, but could only get parts of the data at a time. For this homework, we will just tackle a small part of this problem.

**Question 11: Reading a file**

Make a class called *Lexicon* that when initialized will read in a sorted list of words, stored in *hw2_data.txt*. The filename should be passed in as a parameter. The file is based on [www.ef.edu/english-resources/english-vocabulary/top-3000-words](http://www.ef.edu/english-resources/english-vocabulary/top-3000-words).
Each word consists just of the letters a through z (no capitals, no special characters like `-`). This list of words will not change. Do not use any packages, but simply read it into an array with name `data`, which will be a variable of the class object. Also, include a second object variable called `len` and set this to the number of words in `data`.

Make sure you use ‘with open’ to read the file. Look up on the web how to use ‘with open’ for reading a data file.

**Question 12: Locating an Item**

Create a method called `find_after`, which will take as input a string, and will return the location in the array of the first word that is lexicographically the same or after that string. If there is no such word, it should return `None`.

For example:

```python
l.find_after("motiol") => return location of "motion"
l.find_after("motion") => return location of "motion"
```

This routine should be fast, so make sure it can be done in time \( \log(n) \). Use binary search, and base it on the Pseudo code that is given in the textbook (page 799). (Your version will not have \( p \) and \( r \) passed in, so you will need to compute the initial values for low and high directly.)

Change the code so that if the word is after the last word in the lexicon, it returns `None`.

Verify that your code is able to correctly find every word in the dictionary. If a word is not in the dictionary, is the routine giving you the following word or the preceding word? Check this for every possible word to determine if the code does this consistently and correctly. You can do this by taking every word in the dictionary (except the last one) and adding “aaaaa” to it, and checking if it returns you the index of the next word (assuming it was not the last word).

Comment on what the original code returns when the item is not in the lexicon, turn in your code for testing it, and turn in your code for `find_after`.

**Question 13: Next5**

Make a method called `next5` that will take a string as input, and will return the next words in the table (up to at most 5) that are equal or after the string in lexicographical order. If there are no words that can be returned, it should return an empty list (not `None`).

Here is an example:

```python
l.next5('retirement') => ['retirement','return','reveal','revenue','review']
l.next5('yea') => ['yeah','year','yell','yellow','yes']
```

Turn in your code.

**Question 14: Prefix5**

Make a method called `prefix5` that will take a string as input, and will return the next words in the table (up to at most 5) that start with the input string. If there are no words that can be returned, it should return an empty list (not `None`).

Here is an example:

```python
l.prefix5('retirement') => ['retirement','return','reveal','revenue','review']
```

Turn in your code.
1. prefix5('yea') => ['yeah', 'year']

Turn in your code.

**Question 15: Retrieving with next5**

Using the next5 method, write a function (not a method) that will take the lexicon object as an input, and retrieve all of the data from the lexicon and print each one. You can test if your algorithm is correct if your output is exactly the same as input data file. This version is very straight-forward, unlike the next question.

Turn in your code.

**Question 16: Retrieving with prefix5**

Now assume that the only way to retrieve data from the lexicon is with the prefix5 method, and that the next5 method is not available. Write out in a paragraph how you might go about this.