CSC148 Lab#8, winter 2015

learning goals

In this lab you will explore **BST** methods **insert** and **delete**, both of which may modify the **BST** instance they are called on.

set-up

Open file bst.py in Wing, and save it under a new sub-directory called **lab08**. This file declares classes **BTNode** and **BST**, which implement binary trees and binary search trees, respectively.

You should also save sample1.txt, which has some sample text to experiment with. Recall that a binary search tree is a binary tree where:

- All data are comparable.
- For every node, data in its left sub-tree is less than the data in that node.
- For every node, data in its right sub-tree is less than the data in that node.

experiment with insert

Look at the **if __name__== '__main__':** block at the end of **bst.py**. In that block:

- Write some code to insert the values of **number_list** into a binary search tree. Find the height of the resulting tree using the **height** method. What can you conclude about the performance of the **contains** method?

  Can you think of a way to improve your code so that a more efficient **BST** results? You may want to look at the **random** module.

- Write some code to insert the strings from **word_list** into a binary search tree. Try out **height** and **contains** on the resulting tree. Can you improve the height in some way?

What can you conclude about the impact of the structure of binary search trees on their efficiency? In courses such as CSC263 you will see various ways to approach this question.

experiment with delete

The delete method we developed in class is a bit one-sided: repeated deletions will tend to create a tree where every node has less data in its right sub-tree than its left.

Develop a second method, called **balanced_delete**, that will tend to make a similar number of deletions from the right and left sub-trees of nodes. You may want to look at the **random** module. You will need to write the complete function — docstring and all.