Module 3 Application Exercise

4.7.42)

Since you can have two dogs with the same age it is not possible to generate a perfectly balanced BST. For the dupe handling we will just have it process / structure as if it was an older dog. Each node on the tree will represent a dog to be served on a page, the key attribute here obviously being the age. The initial rendering of the home page will be responsible for generating the initial tree and for every dog adopted an attempt at balancing will occur if necessary.

Will assume we are given a website (which is already in BST form) T which will act as our root node and has attributes called age (self explanatory), leftDog which represents the left node and rightDog which represents the right node. Will also assume there exists a function linkUrlToNewDog to handle the linking for the url links since I don't think that detail matters for this exercise.

For add dog the following must be checked:

1.) if current dog and dog to add nodes are equal and current dog right node is null add to current dog right else traverse right

2.) if current dog age is less than dog to add age and right node is not null traverse right else add to right side

3.) if current dog age is greater than dog to add age and left node is not null traverse left else add to left side

4.) If all dogs have been adopted we will have a null node and say sorry no dogs to adopt (edge case). Given that we should never traverse to a null node since we are checking children at all times the only time this should occur is on initial root check.

```
// start code for addDog

def addDog(currentDog, dogNodeToAdd):
        if currentDog is null:
                // should only occur if root passed in is null
                return “Sorry no dogs left to adopt.”
        if currentDog.age <= dogNodeToAdd.age
                if currentDog.rightDog is null:
                        currentDog.rightDog ← dogNodeToAdd
                        linkUrlToNewDog(currentDog, dogNodeToAdd)
                else:
                        addDog(currentDog.rightDog, dogNodeToAdd)
        elif currentDog.age  > dogNodeToAdd.age:
```

```
            if currentDog.leftDog is null:
                currentDog.leftDog ← dogNodeToAdd
                linkUrlToNewDog(currentDog, dogNodeToAdd)
            else:
                addDog(currentDog.leftDog, dogNodeToAdd)

addDog(T, exampleDogNode)
```

Time complexity: O(log n)

For removeDog we have a little extra work since we need to change node links on removal if a dog that gets adopted is not a leaf node. For this type of handling we will just take the largest left child node and replace the target node. Since we can have duplicate ages in the tree we will leave out rebalancing.

```
// start code for removeDog
def removeDog(currentDog, dogNodeToDelete):
    if currentDog == dogNodeToDelete:
        if currentDog is external dog:
            currentDog ← null
        if currentDog.leftDog and currentDog.rightDog:
            currentDog ← findOldestDogtoLeft(currentDog)
            return currentDog
        elif currentDog.leftDog is null:
            currentDog ← currentDog.rightDog
            return currentDog
        elif currentDog.rightDog is null:
            currentDog ← currentDog.leftDog
    elif currentDog.age <= dogNodeToDelete.age:
        currentDog ← removeDog(currentDog.rightDog, dogNodeToDelete)
        return currentDog
    else:
        currentDog ← removeDog(currentDog.leftDog, dogNodeToDelete)
        return currentDog

removeDog(T, exampleRemoveDog)
```

Time Complexity: O(h) h being the height of tree