Rushith Singamsetty 2/28/2017 Lab Section 104 Tuesday 2pm – 3:15pm

Testfile4.txt:
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

The reason that I chose this particular example is because the alphabet is in ascending order when written. As a result, a binary search tree, if it were to receive the alphabet as its input, would simply keep on adding the next letter of the alphabet as the right node of the previous letter. This results in a tree that is entirely one sided, large, and linear search time of n. There are also a lot of other downsides to this as well besides just longer wait times for find, remove and insert. AVL tree's on the other hand would perform rotations among the tree nodes as they are being added. This helps create a more balanced tree that still has greater values than the current to the right and smaller values than the current to the left. However, its able to rotate stuff as it's being added making which results in smaller search and remove times as you won't have to travel as far down the tree to find a value possibly. For example, find z would take 25 right movements along the binary search tree where as along the AVL tree, it only takes 4 right movements. Thus, in cases where a binary search tree becomes really one sided, an AVL work tree is able to balance out better than a binary search tree allowing for better performance and a much more balanced tree.

```
Testfile3.txt:                              AVL Tree:
BST:                                        Left links followed = 0
Left links followed = 0                     Right links followed = 0
Right links followed = 0                    Total number of nodes = 13
Total number of nodes = 13                  Single Rotations = 1
Avg. node depth = 3.23077                   Double Rotations = 2
                                            Avg. node depth = 2.23077
   Enter word to lookup > the               AVL Tree:
   Word was found: the                      Left links followed = 0
BST:                                        Right links followed = 1
Left links followed = 1                     Total number of nodes = 13
Right links followed = 2                    Single Rotations = 1
Total number of nodes = 13                  Double Rotations = 2
Avg. node depth = 3.23077                   Avg. node depth = 2.23077

Testfile2.txt:                              AVL Tree:
BST:                                        Left links followed = 0
Left links followed = 0                     Right links followed = 0
Right links followed = 0                    Total number of nodes = 16
Total number of nodes = 16                  Single Rotations = 9
Avg. node depth = 6.0625                    Double Rotations = 0
                                            Avg. node depth = 2.5

      Enter word to lookup > everywhere     AVL Tree:
      Word was found: everywhere            Left links followed = 2
BST:                                        Right links followed = 1
Left links followed = 0                     Total number of nodes = 16
Right links followed = 4                    Single Rotations = 9
Total number of nodes = 16                  Double Rotations = 0
Avg. node depth = 6.0625                    Avg. node depth = 2.5
```

Rushith Singamsetty 2/28/2017 Lab Section 104 Tuesday 2pm – 3:15pm

Testfile1.txt:

```
BST:
Left links followed = 0
Right links followed = 0
Total number of nodes = 19
Avg. node depth = 3.15789
```

```
AVL Tree:
Left links followed = 0
Right links followed = 0
Total number of nodes = 19
Single Rotations = 2
Double Rotations = 2
Avg. node depth = 2.26316
```

```
        Enter word to lookup > solve
        Word was found: solve
BST:
Left links followed = 1
Right links followed = 1
Total number of nodes = 19
Avg. node depth = 3.15789
```

```
AVL Tree:
Left links followed = 0
Right links followed = 0
Total number of nodes = 19
Single Rotations = 2
Double Rotations = 2
Avg. node depth = 2.26316
```

Testfile4.txt

```
BST:
Left links followed = 0
Right links followed = 0
Total number of nodes = 26
Avg. node depth = 12.5
```

```
AVL Tree:
Left links followed = 0
Right links followed = 0
Total number of nodes = 26
Single Rotations = 21
Double Rotations = 0
Avg. node depth = 3
```

```
        Enter word to lookup > z
        Word was found: z
BST:
Left links followed = 0
Right links followed = 25
Total number of nodes = 26
Avg. node depth = 12.5
```

```
AVL Tree:
Left links followed = 0
Right links followed = 4
Total number of nodes = 26
Single Rotations = 21
Double Rotations = 0
Avg. node depth = 3
```

These are the results for the four different simulations or test files that I ended up using when testing the a.out of the binary and avl search tree program that we edited in lab today. This data helps us compare the binary search tree to the AVL tree in many different cases.

5) There are many situations in which AVL trees are more preferable to BST trees. One such situation is one that we explored with testfile4.txt in lab today. If there ever exists a time where you have to store data in a tree, if your data consists of many points where the next point is often greater than the previous (or the less than the previous) such that a majority of the tree is one sided is one situation. When this occurs, a binary search tree ends up having a search time of linear n and remove times of this length as well. On the other hand, AVL trees would reduce this processing time by a lot as they help balance out the tree bringing everything closer to the root but also rotating nodes such that it still operates like a binary tree with values lesser than the current node on the left and greater than the current node on the right. However, by rotating nodes to balance out the tree, you are effectively sorting the tree in a way that allows you to search through it, as well as perform other operations, faster. Even with values that would be added in a binary search tree in a much more balanced way, an AVL tree can perform some rotations that will still help it process faster than a binary tree by making the tree much more balanced. Overall, these I believe are the situations that show AVL trees are more preferable.

6) However, at the cost of getting much better find times, there are many downsides to implementing an AVL tree. One such cost is the fact that insert times and remove times can be worse in an AVL tree compared to a binary search tree. This is the result of the AVL tree having to check every node after an insert or remove so that it can make sure it is getting the full benefit of being a balanced tree by rotating nodes when necessary after an insert or remove. This overall results in worse insert and remove times as a result of having to rotate (maybe even double rotate) and sort after an insert and remove than a binary search tree. There is also the fact that AVL trees can be much harder/longer to code than binary search trees due to being much more intricate to include the ability to sort by rotating or double rotating after an insert or remove. Overall these are the costs that are incurred to achieve the O(log(n)) search times of AVL trees in comparision to binary search trees, longer insert and delete times as well as much more intricate coding process due to having include the rotation code necessary to balance the tree.