Bamphiane Annie Phongphouthai

Bp8qg

5-6:15

Testfile4.txt:

annie bamphiane cat dog elephant frog grapes hay igloo job knicks lube mouse nickel organic pearl quit release snow tower universe victor whistle xane yell zena

Searched for: zena

BST:

Left links followed = 0

Right links followed = 25

Total number of nodes = 26

Avg. node depth =24

AVL Tree:

Left links followed = 0

Right links followed =4

Total number of nodes = 26

Single Rotations =21

Double Rotations = 0

Avg. node depth =5

Testfile1.txt; Searched for: using

BST:

Left links followed = 1

Right links followed = 2

Total number of nodes = 19

Avg. node depth =5

AVL Tree:

Left links followed = 0

Right links followed =1

Total number of nodes = 19

Single Rotations =6

Double Rotations = 2

Avg. node depth =3

Testfile2.txt; searched for: higher

BST:

Left links followed = 0

Right links followed = 7

Total number of nodes = 16

Avg. node depth =11

AVL Tree:

Left links followed = 0

Right links followed =0

Total number of nodes = 16

Single Rotations =9

Double Rotations = 0

Avg. node depth =4

Testfile3.txt; searched for: surreal

BST:

Left links followed = 2

Right links followed = 3

Total number of nodes = 13

Avg. node depth =5

AVL Tree:

Left links followed = 1

Right links followed =3

Total number of nodes = 13

Single Rotations =5

Double Rotations = 2

Avg. node depth =3

This example shows that the AVL trees are better than BST because it takes less link follows to retrieve the word. The BST tree takes 25 right links followed, while the AVL tree takes 4 right links followed. Therefore, AVL trees takes less time to find. Giving the BST tree the worst

scenario will allow us to see why the AVL Tree is better. Hence, making a file that makes the insert happen consecutively, will make the BST tree organize like a single linked list. Since the AVL tree keeps balancing, it will take less time to search for the word because the word will be closer to the root. We see that the avg. node depth is 5 which is much less than the BST. For a BST, words with letters that start later in the alphabet will be furthest away from the root because it was inserted this way. The BST tree need to look at each node during the find to see if it is a word, so it would cause for linear time instead of constant. Therefore, this file that inserts the nodes alphabetically will allow one to see why AVL trees are better than BSTs.

To understand when and why AVL trees are preferable to BSTs one must understand how they work. Both trees organize themselves by comparing each element. If the element is greater than the current parent it will be the right child, if it is less it will become the left child. In this case, it will put the first letter that is later in the alphabet to the right and the earlier one to left. The AVL tree, unlike the BST, will adjust the tree in a way that the height of each node will differ by at most one. AVL trees will look at where the insert happened it will look at the parent nodes and check this property, if it is greater than one it will either do a single or double rotation. BSTs will not check this condition therefore, it can be unbalanced.

Unlike BSTs, AVL trees sacrifice a little runtime when adding or deleting to do the balancing step of rotation. This in turn, will make sure that AVL Trees are balanced and the depth of the tree is not large compared to BSTs. BST will most likely be imbalanced and the time for finding the nodes would be linear. As was seen on the lab, the worst case for find on the AVL tree is O(log n) and for BSTs was O(n). AVL trees are preferred when find time of the nodes is the most important. If the user would like to sacrifice the insert/delete then AVL Trees are useful.

The AVL trees are preferred when you are finding from the tree and not doing any manipulation like inserting. Cost for AVL implementation is the adding/deleting. Since the tree has the property to balance itself whenever there is an insert or delete the tree will check for the height of each node. Then it would need to rotate if the height is greater than one. If one uses the insert/delete more than the find, then BSTs may be more useful than the AVL Trees. The added balancing component for the AVL Trees makes it slower than the BSTs in this manner.

In turn, AVL trees are faster than BSTs when you are finding but is slower when insert/deleting.