HW6 report

Kyle Cooper

Code examples:

Below are three figures demonstrating how the program works. Figure 1, shows the text file before anything is done. Figure 2 shows the output of the inorder traversal of the tree, and figure 3, shows the tree root with its children to demonstrate that the height next too the string is correct.

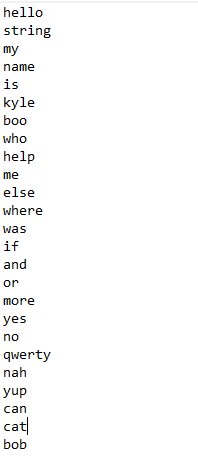
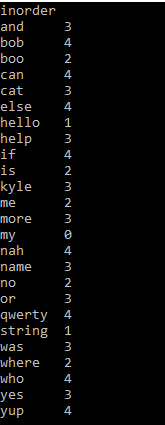
 

Figure 1 Figure 2

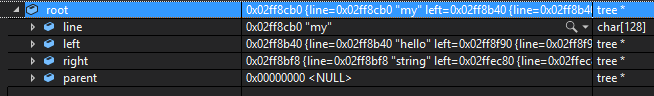


Figure 3

As you can see in Figure 2 and 3. The depth of my is 0, which means it is the root, and the depth of string and hello are 1 which mean they are my’s children.

Below is an additional set of figures 4, 5, and 6, which show a different set of strings, but are the same respectively as figures 1, 2, and above.

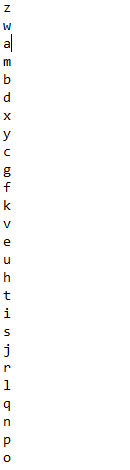
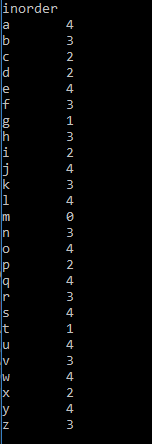
 

Figure 4 Figure 5

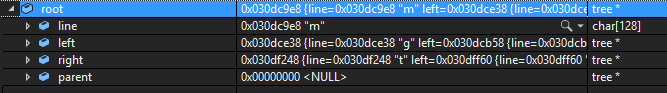


Figure 6

Tree balance algorithm:

Balancing happens right after every insert, and is initiated inside the AVLInsert function, which is based off the AVLInsert pseudocode provided in the slides. After each insert, a check is done on each node from the inserted node up too the root. If a node is found to have an imbalance in it’s children greater than 1, that node is balanced.

This check is done with the if(height(root->left)-1>height(root->right))(there is also another if statement for the other case) compare. I will explain the height function in a moment, but the main thing to notice is the -1 on the root->left side, this is what makes sure that a 1 level difference doesn’t get flagged for rebalancing. Since height will just return the high of the tree from the root given too it.

Below is the height function,

int height(node root) {

int left, right;

if (root == NULL) {

return 0;

}

left = height(root->left);

right = height(root->right);

if (left > right) {

return left+1;

}

if (right > left) {

return right+1;

}

if (right == left) {

return right+1;

}

}

It is a recursive function that finds and returns the height of the tree at the root given to it. It does this by doing a postorder traversal of the tree. If the node it reaches is NULL it will return 0, both as away to give the function a stopping point and so NULL children are not counted in the height.

Say for the tree,

B

/ \

A D

/ \

C F

B is the root, and first it goes to the left subtree, A. Since A is not NULL it goes to A’s left subtree. A’s child is NULL so it returns 0, now A’s right subtree is checked and also returns NULL. Since A’s left and right subtrees are both equal, either right+1 or left+1 can be return both would do the something.

After A returns 1 to the root node B and we know that B’s left subtree has a height of 1, B now checks it’s right subtree, which will end up returning a 2. So we know that B’s right subtree is higher than it’s left subtree, BUT in the if statement at the start we subtract -1 from the compare for the side we added on, so if C was the node added to B, the tree would not be balanced since if(height(root->right)-1>height(root->left)) turns into if(2-1>1) which is not true so the tree is not balanced.

NOTE:

When running my program it just checks for the file text.txt and get the strings from that, if you want to run the program with a specific file, make sure it has the name text.txt