Chris W Jones

CMSC350 – 7380

5 July 2015

# Project 2 Documentation

## Analysis

This week, I started the week by cracking open Introduction to Algorithms (Cormen, 2001). It had quite a bit to say about search trees. I read through the initial chapter about binary search trees and learned how to implement them. I also learned why they are a good structure for searching data.

After I read, I looked over the assignment. I was confused by the starting point for this assignment since the assignment just says BST and doesn’t specify binary or balanced. I thought we were supposed to be using code for a balanced search tree so I found code for a red-black tree and implemented the missing functions. I learned on Saturday after I’d already finished coding that we were just supposed to implement a binary search tree.

My mistake.

## Implementation Details

Even more so than last week, I feel like there wasn’t much for me to design since we were starting from someone else’s code. My biggest decision was if I wanted to use a recursive or iterative approach when it came time to walk the tree. I chose to use recursive methods.

I chose to use recursive methods because, frankly, it was easier. I considered using an iterative approach but it would have been more work. I could easily visualize walking through the tree and which direction to take with the recursive approach. I found it much more difficult to visualize the control logic for an iterative solution.

## User Interface

Once again, I decided to forgo a GUI. I suppose if I wrote a modular GUI, I could reuse it week to week but I don’t see much use for one. Most of our homework assignment haven’t required any user input and those that needed some sort of input could easily use a random number generator to fake user input.

In fact, I wrote a function this week to generate random integers between 0 and 100 so that I could populate the tree. I could have used user input for that but why make it more difficult?

The assignment text mentioned writing a menu for this program. I found a great library for simply generating a command-line interface for programs. I toyed with the idea of using it for this assignment but since we were given a very specific list of functions to call and a specific order to call them in, the only item that my menu would’ve had would be “Run” so I omitted a menu.

## Testing

Testing on this was tough. It was challenging because I didn’t write most of the code so it took me a little bit to figure out it worked. After I determined how the red-black tree code worked, it became easier for me to anticipate the output. I then began to code the methods that I needed to implement: the traversals, height, internal nodes, and leaves.

Determining the correct output of the traversals (pre, post, and in order) was where I did the most testing. It was about this time that I was determining to use an iterative or recursive so I kept rewriting them and checking their output.

To test the height, node, and leaves methods, I had the methods print out the root of the tree and then I wrote out what the tree should look like in a notebook. This gave me a way to check the height of the tree and the leaves. I double checked the number of internal nodes by subtracting the leaves from the total number of nodes (which starts out as 21 on each run).

## Error Handling

In this assignment, I didn’t need to do much error handling. Before I settled on this particular red-black tree implementation, I found one that raised an error when a duplicate value was provided. I found that to be an awful way to handle a duplicate value. I much prefer the way the current code handles a duplicate value which is to basically overwrite the original value with the duplicate value.

Apart from that, the only error handling is checking for null values. That is used in several spots to determine when the end of the tree is reached as well as checking for the root which also has a value of null.

## Conclusions

Binary search trees are really interesting structures. I would have loved to spend a little more time and written a little more code implementing it. If I’d realized that our homework was supposed to use a binary search tree instead of a balanced search tree, I would have coded it myself. Binary search trees seem fairly easy to implement.

Balanced search trees on the other hand are quite hard to write. Because they’re constantly being rearranged to keep them balanced, they have some overhead when inserting or deleting elements. This entails, usually, going through the tree and shuffling the internal nodes around.

I look forward to next week when we start looking at ways to efficiently sort elements of a given data structure.

References

Cormen, T. (2001). *Introduction to algorithms* (2nd ed.). Cambridge, Mass.: MIT Press.