Christian Schmid

Dr. Karla Fant

CS163 - Data Structures

**Program 4 - Write Up**

For Program 4, we are implementing a Binary Search Tree that stores CS terms and definitions from the Carrano reading. The tester for this program will be able to add new terms, edit terms, and remove terms. The tester will also be able to search for provided terms, or a range of terms between two strings.

In this program, I will continue using my CharString class – an ADT that stores and abstracts a char array. This will be the primary data structure for strings in my program. I will use a similar comparison for insertion as used for the HashTable program (which simply adds the ASCII-values of each char together), but the modulus operator won’t be required. When a new node is added to the tree, the CharStrings will be compared. I am also hoping to implement some light parallel programming, using the OpenMP framework.

The BST, while being an efficient and helpful, requires more abstract thinking to solve addition, removal, and retrieval compared to other ADTs – especially considering some of the more nuanced functionality of this program. Addition will be fairly straightforward – simply compare the two CharStrings that represent the new term and the current node’s term. Retrieval will be handled in roughly the same way. Removal will be the most difficult, but using the method discussed in class, it should be fairly straightforward. It is important to note that the nodes of the BST will store pointers to the data, rather than the data itself. This will ensure a quick copying of data when removal requires it. Displaying by range should be straightforward as well – an inorder traversal where each element that falls within the range is displayed.

Finding the height of the tree is the method that I hope to use data parallelism to expedite the process. In theory it seems fairly straightforward: assign a shared integer that first stores -1. When the first leaf is reached, apply its height to the shared int. Each sequential height test will compare its results to the shared int and, if it’s greater than the shared int, the int will take on the new value. After every leaf is reached and tested, return the shared int. The biggest roadblock for creating this function will be determining how to handle when the max thread count is reached. Is it possible to, for instance, wait to process a branch until a thread becomes available? This will be something I will have to research to solve this problem. It is helpful to note that solving the problem this way will make testing the tree’s balance pretty straightforward: simply run the height function, find **2ht – 1** and compare it to the node count.

I think it would be helpful to create a balancer function to, using the array method. Basically I will use the node count to create a dynamic array, which will store all the nodes in order, and then use binary searching re-input the tree. Using pointers to the terms will be helpful in this regard, rather than storing the data itself, for quick allocation and deallocation of the array.

Finally, it would be helpful to describe the traversal orders by which I will run each function. For copying the tree to an array, using inorder traversal will be necessary. For displaying a range of elements inorder traversal will also be best. When deleting the whole tree, postorder will be required, so that both branches will be deleted before the root is.