Jason Klamert

Galina P.

4/23/2016

Project 3 Report

The Relationship between Number of Elements and the Height of Binary Search Trees

To begin with, let me state a formal definition of a binary search tree. A binary search tree, or BST, is a binary tree where each node has a key value and all nodes in the right subtree have key values greater than the parent node. Similarly, all nodes in the left subtree from the parent node have key values less than the parent node. However, this leaves some gray area that is undefined. What happens when we have duplicate key values in a BST? Well, we have two options that are possible. We may place a node with the duplicate key value in either the left or the right subtree from the parent node. In my project I have chosen to place duplicate key values in the left subtree from the parent node. I feel like it is more appropriate to place these nodes in the left subtree over the right subtree and treat it like a ‘less than’ case. The insert function simply compares the key value of the root to the value passed into the function. If the value is less than or equal to root’s key value, then the function checks if there is a left child from the parent. Likewise, if the value is greater than the key value then the function checks if there is a right child. If one of these conditions is true and there is a child then we traverse down the tree in that direction and execute these key value comparisons at every node. We keep traversing in this fashion until we find a null child during traversal. This is our notice to insert our new value into the BST. This series of operations is mirrored for values strictly greater than the parent key value. Now we shall cover the theoretical efficiency of the function getHeight.

The function getHeight returns the height of a node in a BST. I should note that the height is defined as the number of edges from the node to the deepest leaf. Much like the traversal algorithms, I believe that the function getHeight has a time complexity of T(n) = Ɵ(n). I believe that this function has a linear time because we need to visit and every node exactly one time to determine which leaf has the greatest depth. In fact, this function should have the same time complexity as all of the main traversal functions for BSTs. In-order, pre-order, and post-order traversals all have an efficiency of T(n) = Ɵ(n). With this insight we may come up with a recurrence relation. Since we have two decisions at each node, and we obtain a sub problem of half of the original size, During the ‘recombine’ step we are going to assume that it takes constant time, T(n) = (1). Thus, we may obtain the following recurrence: T(n) = 2T(n/2) + Ɵ(1). This is the best case recurrence which features a full binary tree. If we solve this recurrence with the Master Theorem we obtain T(n) = Ɵ(n). In the worst case we get a degenerate binary tree. This recurrence would be described by the following: T(n) = T(n-1) + Ɵ(1) due to a constant ‘recombine’ step. The recurrence would be the one above due to the fact that each node only has one child so our problem only shrinks by one for each call. By using backwards substitution we may solve this recurrence.

T(n) = T(n-1) + C

T(n-1) = T(n-2) + C

T(n) = [T(n-2) + C] + C

T(n-2) = T(n-3) + C

T(n) = [T(n-3) + C] + C + C

We start to see a pattern and we come up with a time complexity that is T(n) = Ɵ(n). The recursion reaches a base case and stops with a constant, or 1, being added ‘n’ times. This concludes our discussion on the theoretical efficiency of the getHeight algorithm.

In conclusion, the measures of height that were taken during this project were averaged over three cases and rounded down to the nearest integer. The ‘T’ in the chart means that ‘T’ binary search trees will be generated for trees of each size. It seems that duplicate values are one of the worst cases for achieving the least possible height. Please look at the chart and find the results of the project. You will find that the case for T = 5 may not be as accurate as the cases where T > 5. It appears that when T = 5 the height is under represented. However, when T = 10 the height appears to be over represented. When T = 15 the two cases seem to be averaged. Overall, when 'T' becomes sufficiently large the height seems to become more accurate. The results for when T = 15 even hold for sufficiently large values of T, even though they are not listed in the results. This concludes the discussion on theoretical efficiency and experimental results.