Jessica Krynitsky

03/02/18

12:30 Section

The output of all three test files and a fourth test indicate that the lookup time is faster for the AVL tree than for the Binary Search Tree. This can be quantified by counting the number of paths followed from the root node to a leaf node when a find operation is performed. These numbers are represented in the tables below. Unfortunately, actual runtimes are not a good measurement for such small numbers of values

|  |  |  |
| --- | --- | --- |
| Testfile1 | | |
|  | Total Number of Nodes | Paths followed to leaf |
| BST | 18 | 6 |
| AVL | 18 | 4 |
|  |  |  |
| Testfile2 | | |
|  | Total Number of Nodes | Paths followed to leaf |
| BST | 14 | 7 |
| AVL | 14 | 4 |
|  |  |  |
| Testfile3 | | |
|  | Total Number of Nodes | Paths followed to leaf |
| BST | 11 | 7 |
| AVL | 11 | 4 |
|  |  |  |
| Test 4 | | |
|  | Total Number of Nodes | Paths followed to leaf |
| BST | 6 | 4 |
| AVL | 6 | 2 |

The largest separator for AVL trees and simple binary search trees is the balance factor. If the difference in heights somewhere in an AVL tree is more than one, then the tree will perform rotations and restructure so that this is not the case. What this does is ensures O(logn) runtime (worst-case). For comparison, the worst case for a binary search tree would be something like if every node inserted was greater than the previous value. This would result in a tree with a long line of right subtrees which is essentially like a linked list. Each of these nodes would have to be examined when searching or removing an element, which would be O(n) (linear) worst case runtime. Based on this, it is easy to see why an AVL tree would be preferred to a binary search tree. However, in most cases the average runtime is the same for AVL and BS trees. AVL trees would be preferred in situations in which frequent lookups are required, but insertion and deletion happen less frequently.

AVL trees are not better than Binary Search trees in every way. One thing to consider is the balance factor: each node in an AVL tree has one extra piece of information to keep in storage. This is dynamically allocated memory, so results in AVL trees taking up more space in memory. Additionally, insertion and deletion may be much slower than for a binary search tree, because the tree must be balanced every time which could mean being rotated. These rotations take up time, so if many nodes are being rapidly inserted into and removed from the tree, a binary search tree may have a faster runtime. Is a fast search time worth the cost of slower insertion and deletion? This is a case by case answer which depends both on the data and on the preference for the program.