DSA I Lab 5: Trees

Christine Baca [cab8xd]

1 University of Virginia, Charlottesville VA 22904, USA

**Abstract.** The experiment focused on implementing the structures of a Binary Search Tree (BST) and an AVL tree. Binary search trees take in values and sorts them into a tree structure of nodes. AVL extends or inherits a BST but rotates and reorganizes the tree to the most balanced structure in order to lower the runtime for search or search-based methods. To compare the structures, data was inserted and manipulated using both BST and AVL class methods and timed using Java’s system. Several trials were conducted using two datasets: one dataset of randomized integers and another as a text file of strings. The experiment concludes that, when inserting and searching for integers or strings, the BST tree holds a faster average runtime and lower standard deviation than the AVL tree, possibly because its simpler code and that that the complexity of the AVL balance method may have slowed down the AVL tree’s overall runtimes or the size of the datasets used for testing.

**Keywords:** trees, BST, AVL Trees, Java, nodes, height, balancing, search, insert

1. Introduction

The lab explored the efficiency of a BST tree data structure in comparison to an AVL tree data structure implemented in Java using nodes by inserting and manipulating a large data set using each class’s methods.

* 1. BST

A BST (binary search tree) takes in nodes and sorts them in a tree-structure where each node branches out from its parent or root, and are sorted by comparison; every right child node or subtree references a value or values greater than its parents’ or roots’ reference value. The code below shows a sample test of a String-type binary search tree and its insert method.

//Starting timer

t = (**int**) System.*currentTimeMillis*();

//Adding string words to to BST

**for**(**int** i=0; i<s.size(); i++) {

bst.insert(s.get(i));

}

//Stopping timer

time = ((**int**) System.*currentTimeMillis*()) - t;

//Recording height

tXBST.add(time);

//Recording height

hXBST.add(bst.height())

* 1. AVL Trees

AVL inherits the BST data structure and implements implicit rotation methods that adjusts the balance factors and node position within the tree to make the structure more efficient in methods that require finding an element. The code below shows a sample test of an integer-type AVL tree and its search method.

//Find method

**for**(**int** j = 0; j<numElements; j++) {

**if**(!avl2.find(toAdd.get(j))) {

System.***out***.println("FATAL ERROR: Element " + j + " was not found in your tree!");

System.*exit*(1);

}

}

//timing avl

time = ((**int**) System.*currentTimeMillis*()) - t;

tRAVLs.add(time);

//adding height

hRAVLs.add(avl2.height());

* 1. Test Code & Predictions

The predicted outcome of the experiment is that the AVL methods overall will illustrate a faster or similar average runtimes than BST because the AVL tree is the improved upon version of the BST that sorts the tree to prevent edge cases that would cause slow runtimes in a BST tree. The average deviation for the AVL methods should be less than the BST method runtimes because the balanced subtrees of the AVL tree should equalize the search runtimes of its individual subtrees in contrast to the unbalanced structure that typically occurs in a BST.

The BST and AVL tree methods insert and search will be tested with data sets of random integers and random string words.

1. Experiment:

The procedure for the experiment runs through numerous trials of the BST and AVL tree methods using both a pre-selected and randomized data sets, recording the runtime of the methods and the height of the trees. The average and standard deviation of the trials will be calculated and recorded. The data will be analyzed based on the trial’s tree height along with runtimes, averages, and standard deviations of the runtimes. The experiment table lists the number of trials, runs, and additional information regarding the experimental procedure.

* 1. Experiment Table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **addNum Insert** | **addNum Search** | **addString Insert** | **addString Search** |
| **No. of Trials (per Tree & Method)** | 15 | 15 | 15 | 15 |
| **Number of Random Numbers/Words inserted/searched within Tree** | 50,000 | 50,000 | 24,000 | 24,000 |
| **Total No. of Method runs** | 750,000 | 750,000 | 150,0000 | 150,0000 |

1. Results
   1. Table Results

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Trials | addNum: BST insert | addNum: AVL insert | addNum: BST search | addNum: AVL search | addString: BST insert | addString: AVL insert | addString: BST search | addString: AVL search |
| 1 | 2383 | 3125 | 386 | 5135 | 157 | 284 | 386 | 489 |
| 2 | 2568 | 3182 | 409 | 5512 | 145 | 292 | 409 | 518 |
| 3 | 2284 | 3746 | 294 | 6027 | 110 | 214 | 294 | 374 |
| 4 | 3281 | 3863 | 467 | 5996 | 145 | 280 | 467 | 572 |
| 5 | 2044 | 2852 | 313 | 4577 | 102 | 208 | 313 | 419 |
| 6 | 2220 | 3259 | 291 | 6176 | 103 | 337 | 291 | 373 |
| 7 | 3164 | 4939 | 507 | 8382 | 117 | 613 | 507 | 670 |
| 8 | 4231 | 5391 | 788 | 9019 | 209 | 415 | 788 | 955 |
| 9 | 3765 | 4776 | 599 | 8225 | 197 | 265 | 599 | 752 |
| 10 | 2477 | 3269 | 362 | 5811 | 140 | 345 | 362 | 452 |
| 11 | 2952 | 4043 | 359 | 7669 | 128 | 399 | 359 | 448 |
| 12 | 3635 | 4648 | 493 | 8304 | 175 | 376 | 493 | 628 |
| 13 | 4025 | 5074 | 560 | 8772 | 199 | 232 | 560 | 721 |
| 14 | 3600 | 4645 | 526 | 6649 | 193 | 262 | 526 | 660 |
| 15 | 2375 | 3300 | 324 | 6996 | 111 | 210 | 324 | 409 |
| Average | 3000.3 | 4007.5 | 148.7 | 6883.3 | 148.73 | 315.4 | 445.2 | 562.6 |
| Standard Deviation | 721.6 | 837.5 | 37.7 | 1421.87 | 37.74 | 106.4 | 137.6 | 166.6 |
| Average Height | 887 | 22 | - | - | 39 | 18 | - | - |

3.2 Graph Results

1. Conclusion

The experiment tested the runtimes between a BST and AVL tree’s insert and search methods. Several trials were conducted using two datasets: one dataset of randomized integers and another as a text file of strings. The experiment concludes that, when inserting and searching for integers or strings, the BST tree holds a faster average runtime and lower standard deviation than the AVL tree. Both trees show a linear runtime, but AVL trees deny any inconveniently shaped trees from being organized and theoretically should have a more consistent runtime. This conclusion contradicts the prediction of the AVL tree having the faster runtimes because of its implicit balance method and shorter height. The faster runtime of the BST may be due to its simpler code and the balance method may have slowed down the AVL tree’s runtime or the size of the datasets. And, the string methods were processed significantly faster than the integer methods for both trees, possibly due to the nature of string comparisons versus integer comparisons. Errors such as faulty code or testing may have skewed the data results. Further experiments would test other implementations of insert and search for the BST and AVL tree to see if the runtimes would differ.