Report AVL vs. BST

Author: Bilal Hasan Aslan

02.03.2019

Bilalhasanaslan@gmail.com

CONTENTS PAGE

Bilalhasanaslan@gmail.com

«EXPERIMENT»

Aim

Aim of this project is to test efficiency of searching and inserting in Binary search tree when we compare it to the searching and inserting to AVL Tree.

Method

We are going to read csv file that is provided. Then we are going to insert data from this csv file to both AVL Tree and Binary Search Tree. While inserting we will gather number of comparisons made when inserting to compare their effectiveness for inserting. Then we going to search keys though out stored data in trees and gather number of comparisons made while searching again to compare effectiveness for searching. We will get best, average and worst case for each search and draw graph from gathered counts to make a conclusion about their efficiencies.

So we need Insert, Search, getTotalCount and testing methods for both trees.

Classes

We Create two different java classes where one for each experiment:

1. **PowerAVLApp** – This java class is for testing search and insert for AVL tree concept and also contains some other methods.
   Main() : Reads csv file with scanner class and stores it in Nodes with AVL Tree concept.
   testing() : This method can be used for testing efficiency of inserting and searching in the AVL Tree. It will write data to txt file called “DATA”. This data will be used further in experiment.

Inner classes for PowerAVLApp*:*

AVLTree –

InnerClass ^ Node(String d) : Basic node class.

BinarySearchTree() : Makes Node root null.

zeroCount() : equates Counts that are used for testing to zero.

height(Node N) : finds height of AVL Tree that will be later used to balance the tree.

max(int a, int b) : return bigger number between a and b.

rightRotate(Node y) : A utility method to right rotate subtree rooted with y.

leftRotate(Node x) : A utility function to left rotate subtree rooted with x.

getBalance(Node N) : Get Balance height factor of node N.

insert(String key) **:** mainly calls **insert(Node root, String key)** method.

insert(Node root,String key) **:** with given key we go through node called root with comparing and recurring to find perfect place to add and store this key in node root. After inserting we check heights of the nodes and if they are not balanced, we balance them.

search(String key) : mainly calls **search(Node root, String key).** method.

search(Node root, String Key) : with given Node root this method compares and searches through Nodes recursively to find given key.

inOrder() : mainly calls inorderRec method.

inorderRec(Node root) : Prints data from Node root with the order its stored.

writeToFile() : writes Count numbers to a file called “countAVL.txt”.

getTotalCount() : Adds and returns, insert and search Count numbers.

(this inner class is written with the help from https://www.geeksforgeeks.org/avl-tree-set-1-insertion/)*

2. **PowerBSTApp** – This java class is for testing search for Binary search tree concept and also contains some other methods.

   Main() : Reads csv file with scanner class and stores it in Nodes with Binary search tree concept.

   Testing() : This method can be used for testing efficiency of inserting and searching in the BST Tree. It will write data to txt file called "DATA2". This data will be used further in experiment.

   Inner classes for PowerBSTApp:

   BinarySearchTree –

   InnerClass ^ Node(String key) : Basic node class.

   BinarySearchTree() : Makes Node root null.

   insert(String key): mainly calls insertRec method.

   insertRec (Node root,String key): with given key we go through node called root with comparing and recurring to find perfect place to add and store this key in node root.

   search(Node root, String Key) : with given Node root this method compares and searches through Nodes recursively to find given key.

   inOrder() : mainly calls inorderRec method.

   inorderRec(Node root) : Prints data from Node root with the order its stored.

   getTotalCount() : Adds and returns, insert and search Count numbers.

   writeToFile() : writes Count numbers to a file called "countBST.txt".

   (this inner class is written with the help from https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion)*

Bilalhasanaslan@gmail.com

«TEST VALUES AND OTHER OUTPUTS»

Part 2

For this part we are going to use PowerAVLApp class.

1. For Date : 17/12/2006/17:43:00 We get Count : 11
2. For Date : 16/12/2006/17:24:00 We get Count : 1
3. For Date : 16/12/2006/17:29:00 We get Count : 3
4. With unknown date we get “Date/time not found”.

First 10 Data:

```
16/12/2006/17:24:00, 4.216, 234.840
16/12/2006/17:25:00, 5.360, 233.630
16/12/2006/17:26:00, 5.374, 233.290
16/12/2006/17:27:00, 5.388, 233.740
16/12/2006/17:28:00, 3.666, 235.680
16/12/2006/17:29:00, 3.520, 235.020
16/12/2006/17:30:00, 3.702, 235.090
16/12/2006/17:31:00, 3.700, 235.220
16/12/2006/17:32:00, 3.668, 233.990
16/12/2006/17:33:00, 3.662, 233.860
```

Last 10 Data:

```
17/12/2006/01:34:00, 2.358, 241.540
17/12/2006/01:35:00, 3.954, 239.840
17/12/2006/01:36:00, 3.746, 240.360
17/12/2006/01:37:00, 3.944, 239.790
17/12/2006/01:38:00, 3.680, 239.550
17/12/2006/01:39:00, 1.670, 242.210
17/12/2006/01:40:00, 3.214, 241.920
17/12/2006/01:41:00, 4.500, 240.420
17/12/2006/01:42:00, 3.800, 241.780
17/12/2006/01:43:00, 2.664, 243.310
```

Part 4

For this part we are going to use PowerBSTApp class.

1. For Date : 16/12/2006/17:43:00 We get Count : 18
2. For Date : 16/12/2006/19:51:00 We get Count : 1
3. For Date : 17/12/2006/00:29:00 We get Count : 3
4. With unknown date we get “Date/time not found”.

First 10 Data:

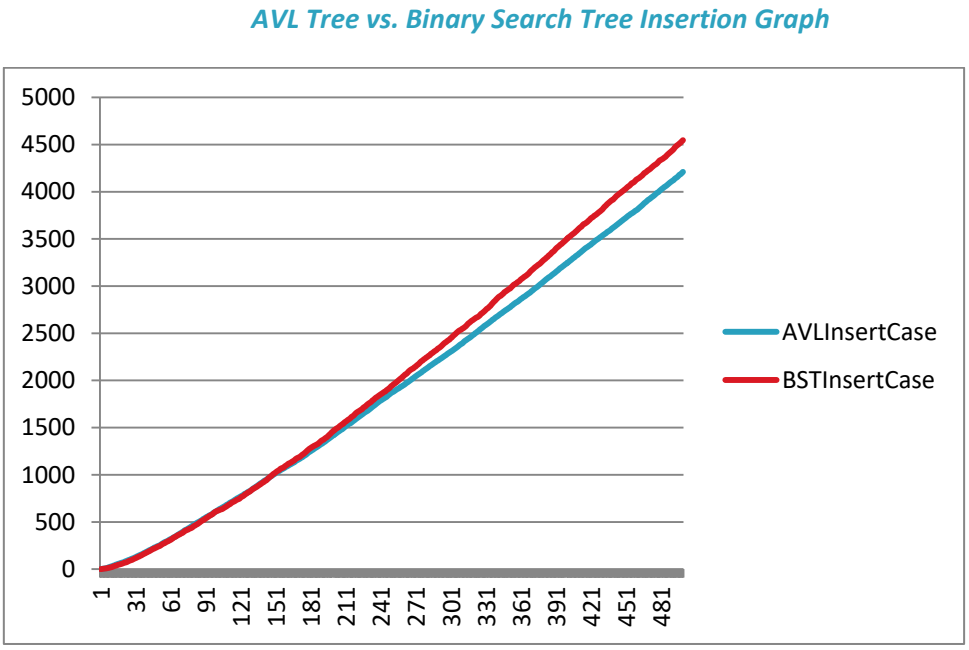
```
16/12/2006/17:24:00, 4.216, 234.840
16/12/2006/17:25:00, 5.360, 233.630
16/12/2006/17:26:00, 5.374, 233.290
16/12/2006/17:27:00, 5.388, 233.740
16/12/2006/17:28:00, 3.666, 235.680
16/12/2006/17:29:00, 3.520, 235.020
16/12/2006/17:30:00, 3.702, 235.090
16/12/2006/17:31:00, 3.700, 235.220
16/12/2006/17:32:00, 3.668, 233.990
16/12/2006/17:33:00, 3.662, 233.860
```

Last 10 Data:

```
17/12/2006/01:34:00, 2.358, 241.540
17/12/2006/01:35:00, 3.954, 239.840
17/12/2006/01:36:00, 3.746, 240.360
17/12/2006/01:37:00, 3.944, 239.790
17/12/2006/01:38:00, 3.680, 239.550
17/12/2006/01:39:00, 1.670, 242.210
17/12/2006/01:40:00, 3.214, 241.920
17/12/2006/01:41:00, 4.500, 240.420
17/12/2006/01:42:00, 3.800, 241.780
17/12/2006/01:43:00, 2.664, 243.310
```

Bilalhasanaslan@gmail.com

«Graphs for unsorted Data»

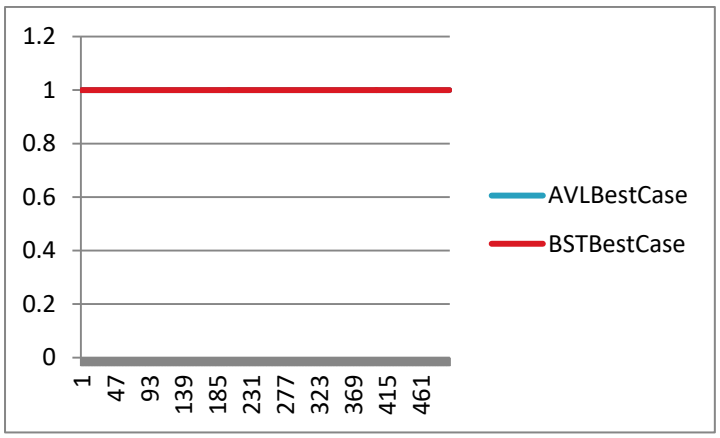


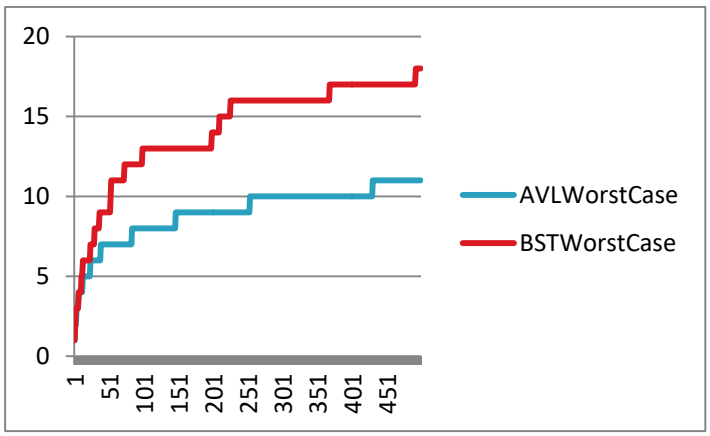
About Graph

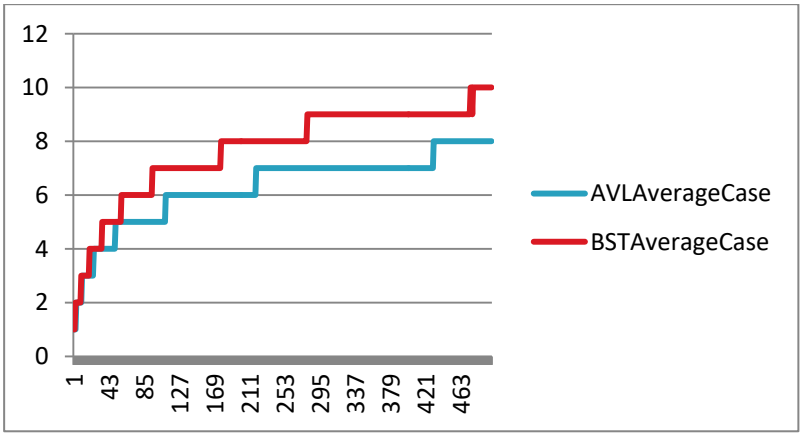
In Creating of this graph, Code is written so it will read txt file line by line, insert dates to AVL or BST one by one and write how many times compare method is used, into a text file. We can see that AVL insertion is slightly more efficient then BST insertion for this unsorted Data.

Bilalhasanaslan@gmail.com

AVL Tree vs. Binary Search Tree Best, Average and Worst Case Graphs







About Graphs

In Creating of these graphs, Code is written so it will read txt file line by line, insert dates to AVL or BST one by one then search every key that is inserted and write how many times compare method is used while searching, into a text file. We can see that AVL insertion is more efficient then BST insertion for this unsorted Data.

Analyzing Best, Average and Worst from Graphs

We can conclude the table underneath from graphs but only not the worst scenario for Binary Search Tree concept because with given data we didn't experience it. We will see worst case for Binary Search Tree with Sorted Data.

Search Method	AVL Tree	BinarySearchTree
Best	O(1)	O(1)
Average	O(log n)	O(log n)
Worst	O(log n)	*O(n)

Analyzing All Graphs Together

To decrease search count gradually we use these AVL Tree and BST Concept. When we compare their insertion method BST is slightly slower than AVL Tree, that makes AVL is more efficient than BST in insertion. When we compare their search method again BST is slightly slower than AVL Tree where as they both follow O(log n) time complexity.

«About Sorted Data»

We get completely different results when we sort the data before inserting it to BST. Table below made with 500 data that is sorted.

Search Method	AVL Tree	BinarySearchTree
Best	1	1
Average	7	250
Worst	9	500
Insert Method	-	-
Insertion	4480	124750

Analyzing Table

From the table above we can see that BST is linear with very huge insertion count number while AVL Tree is even faster than its unsorted data counts. AVL Tree keeps balance of the height of the tree to make it more efficient but BST will be just linear.

«Conclusion»

Results

As we can see from table and graphs, inserting and searching with AVL Tree concept is more efficient when it is compared to inserting and searching in Binary Search Tree concept. With sorted data AVL Tree still keeps its efficiency while BST becomes very inefficient.

When it comes to very large data sets it is much more efficient to use search method with AVL Tree. Whereas coding for AVL tree is more complicated to BST.

Extras (Creativity)

Method testing is made for checking insert and search efficiency to get efficiency information to create a graph.

Method writeToFile is made to write numbers of counts of comparisons are made while inserting and searching, to a file.

Method getTotalCount is made to get total number of counts of comparisons are made while inserting and searching.

Read me File is provided to show how to use MakeFile

Git Log

«REFERENCES»

https://www.geeksforgeeks.org/binary-search-tree-set-1-search-and-insertion

https://www.geeksforgeeks.org/avl-tree-set-1-insertion/

Vula Lecture Slides.