CSC2001 ASSIGNMENT 2

DATA STRUCTURES: AVL TREE

Yaseen Hull
HLLYAS001

Contents

What is the problem?

AVL data structures are implemented by adding certain conditions to ordinary Binary Search Trees to improve performance. These conditions entail keeping a record of the height of the tree to achieve a type of balanced tree. We can assume that every node in an AVL Tree has a key, data, height, left and right and that keys can be compared either by string or numerically. The problem is for the AVL algorithm to verify that the tree is; a Binary Search Tree, the height information stored in each node is correct and that every node is balanced. Hence it must show that after each node insertion the tree must satisfy these three conditions.

The task aim is to explore the efficiency of the AVL Search Tree data structure. By comparing a traditional Binary Search Tree search operation, the speed and memory used in executing tasks can be identified. The task is therefore to write an electronic phonebook system in java by implementing a AVL Search Tree data structure to keep record of entries and perform various operations on the data held by the database. To test out the AVL Search Tree one main operation must be developed known as the SearchAVL application. Theoretically the application must traverse through the data structure until the electronic phone entries are found.

Design

Before either of applications can run the test data provided must be concatenated to just the name of a person of the entry. The name will therefore act as a key and hence it is the main value in determining the output of the two methods. The efficiency will be assessed based on time complexity of the search functions i.e. the time taken for several entries n to be printed to the screen.

The design comprises of two main Operations both of which traverses though the Binary Search Tree structures to find and print the information contained within nodes. The tree nodes of both data structures are linked to the root/ head node and are connected to either right or left branch of the root depending on its value. The value is determined by the key and in the case of the electronic phone book the key is a string name of the contact entry. Hence the structure is formed with respect to the alphabetical order after the root node is inserted. If the entry after the root node is inserted has a key which is positioned alphabetically higher than the root key then it is inserted in the right branch. Else if it is positioned alphabetically lower then it is linked to the left branch. Furthermore, if an entries key is higher chronologically than the root but lower than the parent in the right sub-tree it is then positioned on the left branch of the right parent node. A similar conclusion can be drawn from an entry with a key value lower than the root key but higher than the left sub-tree. The two methods designed for this application has been named SearchAVL and SearchIt and are discussed below.

SearchIt method

The 'SearchIt' method requires a database of entries to exist to be operational. However, if the root is null the search method would just return null. The 'SearchIt' method requires an input value to compare with. In the case of the phonebook the key was set as the name of the contact which was also the input for the search method. A query file was created containing just the names of the contacts in the phonebook to use as a comparison. If the root was not null the string name and the node root were taken as arguments for the method. Three cases existed for this comparison. The first checked if the name input was the same as the key of the root. The second checked if the name input was chronologically lower than the root key and the last checked if it was chronologically higher than the root key. Recursion had to be taken into consideration as

The first case of the search method compares a String name and the key from the node starting at the root. If they're the same the node entry (contact information) is returned and printed to the screen. If this isn't the case the next condition is initiated.

The next two cases depend on the value of the inserted String. Either the string is higher or lower than the root node. If this is the case the left subtree or right subtree must be traversed to the next node to find the key which matches the inserted string. If either key of the left and right subtree nodes fails to match the method of search is invoked again until the first condition is satisfied.

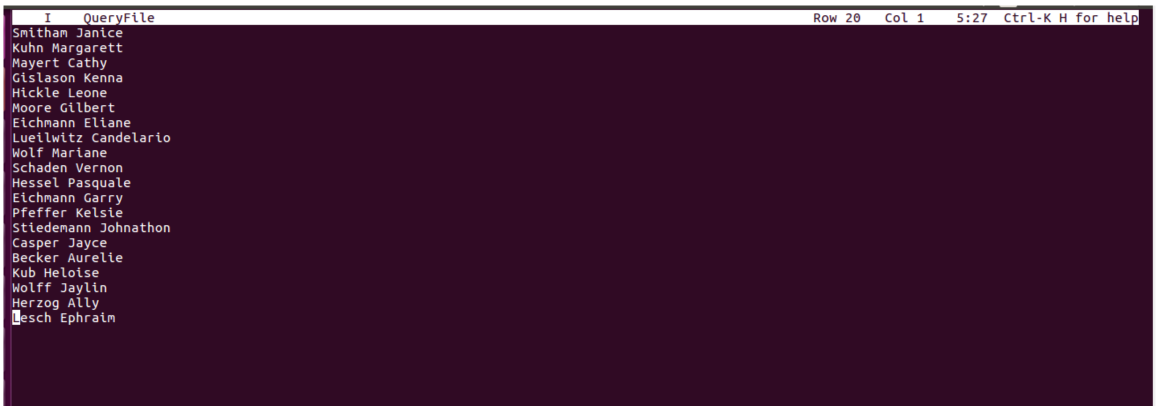
SearchAVL method

The SearchAVL application is a much more efficient manner of traversing the tree. Its success lies in the fact that instead of a O(n) from an ordinary BST it has a 0(logn) time complexity. The cause is due to its structure which adheres to a few conditions upon insertion of nodes. The SearchAVL application manages to maintain a balanced tree by adjusting the structure (if need be) every time a new entry is inserted. Its conditions are that for every node the left and right subtrees may not exceed a height difference of one. This means that search times are boosted when a search operation is invoked because binary search times is cut in half each time a level in the data structure is descended.

The SearchAVL application is basically a balanced BST hence its search methods are the same as the Search SearchIt explained previously. As I said before the time complexity for the worse case in a AVL Tree is O(logn) thus making it more efficient than the Binary Search Tree.

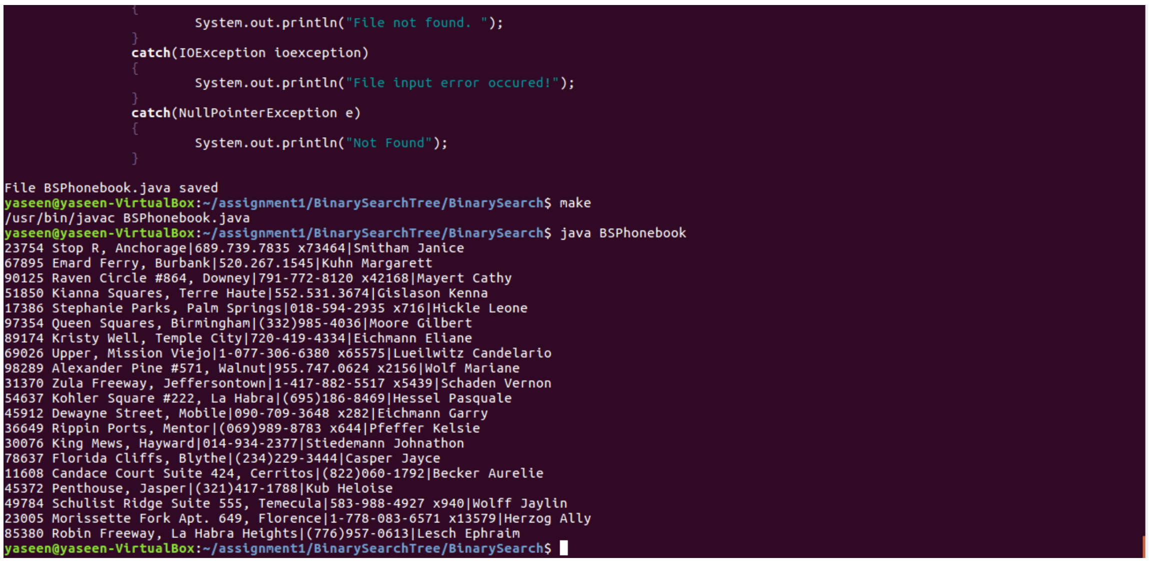
Code input and outputs

QueryFile input

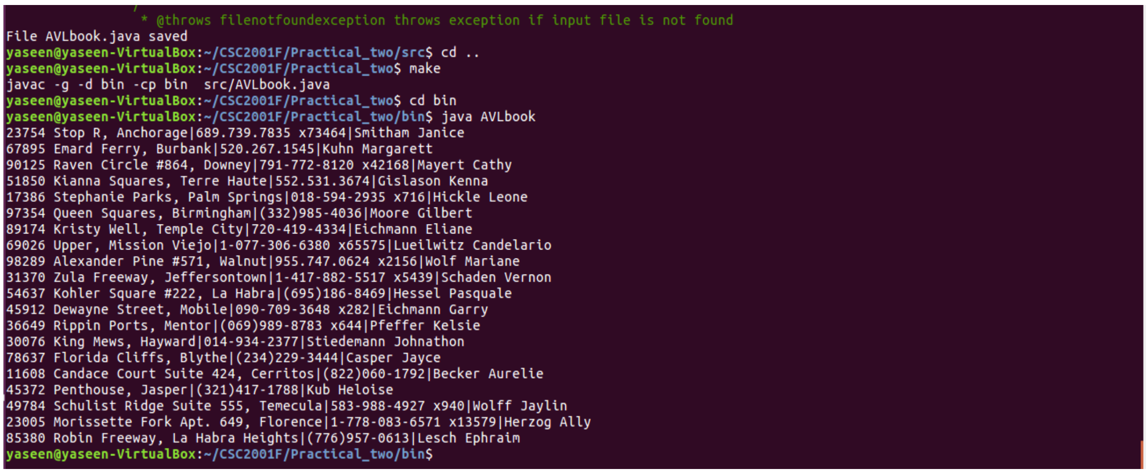


```
  I    QueryFile                                                                                Row 20    Col 1     5:27  Ctrl-K H for help
Smitham Janice
Kuhn Margarett
Mayert Cathy
Gislason Kenna
Hickle Leone
Moore Gilbert
Eichmann Eliane
Lueilwitz Candelario
Wolf Mariane
Schaden Vernon
Hessel Pasquale
Eichmann Garry
Pfeffer Kelsie
Stiedemann Johnathon
Casper Jayce
Becker Aurelie
Kub Heloise
Wolff Jaylin
Herzog Ally
Lesch Ephraim
```

SearchIt and SearchAVL output



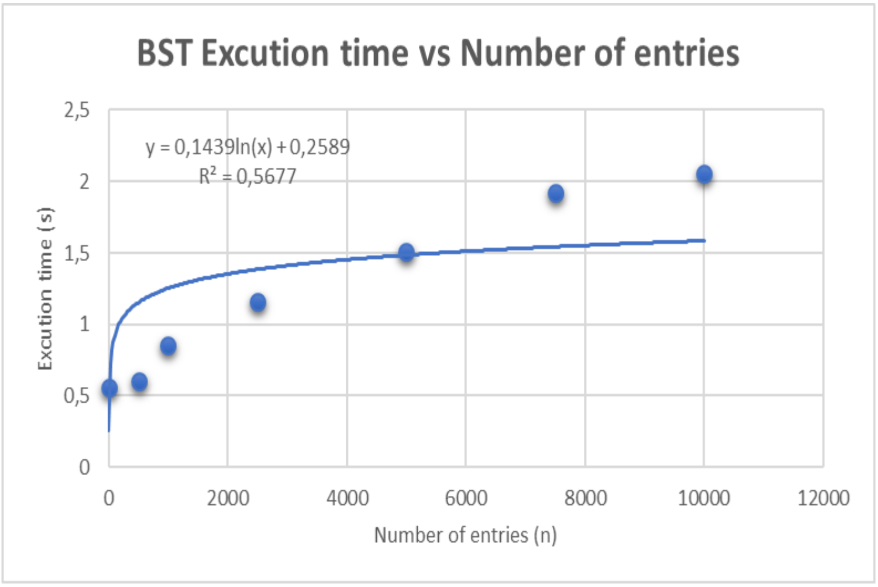
```
                {
                        System.out.println("File not found. ");
                }
                catch(IOException ioexception)
                {
                        System.out.println("File input error occured!");
                }
                catch(NullPointerException e)
                {
                        System.out.println("Not Found");
                }

File BSPhonebook.java saved
yaseen@yaseen-VirtualBox:~/assignment1/BinarySearchTree/BinarySearch$ make
/usr/bin/javac BSPhonebook.java
yaseen@yaseen-VirtualBox:~/assignment1/BinarySearchTree/BinarySearch$ java BSPhonebook
23754 Stop R, Anchorage|689.739.7835 x73464|Smitham Janice
67895 Emard Ferry, Burbank|520.267.1545|Kuhn Margarett
90125 Raven Circle #864, Downey|791-772-8120 x42168|Mayert Cathy
51850 Kianna Squares, Terre Haute|552.531.3674|Gislason Kenna
17386 Stephanie Parks, Palm Springs|018-594-2935 x716|Hickle Leone
97354 Queen Squares, Birmingham|(332)985-4036|Moore Gilbert
89174 Kristy Well, Temple City|720-419-4334|Eichmann Eliane
69026 Upper, Mission Viejo|1-077-306-6380 x65575|Lueilwitz Candelario
98289 Alexander Pine #571, Walnut|955.747.0624 x2156|Wolf Mariane
31370 Zula Freeway, Jeffersontown|1-417-882-5517 x5439|Schaden Vernon
54637 Kohler Square #222, La Habra|(695)186-8469|Hessel Pasquale
45912 Dewayne Street, Mobile|090-709-3648 x282|Eichmann Garry
36649 Rippin Ports, Mentor|(069)989-8783 x644|Pfeffer Kelsie
30076 King Mews, Hayward|014-934-2377|Stiedemann Johnathon
78637 Florida Cliffs, Blythe|(234)229-3444|Casper Jayce
11608 Candace Court Suite 424, Cerritos|(822)060-1792|Becker Aurelie
45372 Penthouse, Jasper|(321)417-1788|Kub Heloise
49784 Schulist Ridge Suite 555, Temecula|583-988-4927 x940|Wolff Jaylin
23005 Morissette Fork Apt. 649, Florence|1-778-083-6571 x13579|Herzog Ally
85380 Robin Freeway, La Habra Heights|(776)957-0613|Lesch Ephraim
yaseen@yaseen-VirtualBox:~/assignment1/BinarySearchTree/BinarySearch$
```



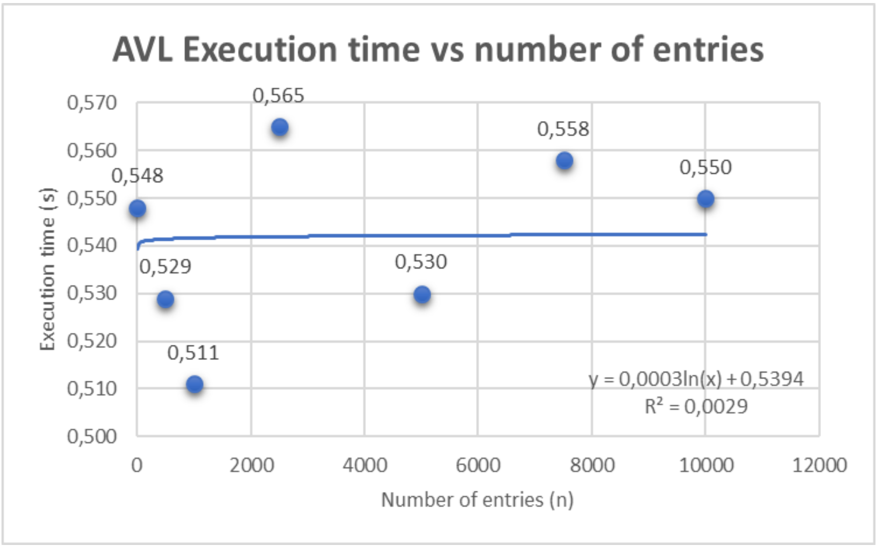
```
                /
                 * @throws filenotfoundexception throws exception if input file is not found
File AVLbook.java saved
yaseen@yaseen-VirtualBox:~/CSC2001F/Practical_two/src$ cd ..
yaseen@yaseen-VirtualBox:~/CSC2001F/Practical_two$ make
javac -g -d bin -cp bin  src/AVLbook.java
yaseen@yaseen-VirtualBox:~/CSC2001F/Practical_two$ cd bin
yaseen@yaseen-VirtualBox:~/CSC2001F/Practical_two/bin$ java AVLbook
23754 Stop R, Anchorage|689.739.7835 x73464|Smitham Janice
67895 Emard Ferry, Burbank|520.267.1545|Kuhn Margarett
90125 Raven Circle #864, Downey|791-772-8120 x42168|Mayert Cathy
51850 Kianna Squares, Terre Haute|552.531.3674|Gislason Kenna
17386 Stephanie Parks, Palm Springs|018-594-2935 x716|Hickle Leone
97354 Queen Squares, Birmingham|(332)985-4036|Moore Gilbert
89174 Kristy Well, Temple City|720-419-4334|Eichmann Eliane
69026 Upper, Mission Viejo|1-077-306-6380 x65575|Lueilwitz Candelario
98289 Alexander Pine #571, Walnut|955.747.0624 x2156|Wolf Mariane
31370 Zula Freeway, Jeffersontown|1-417-882-5517 x5439|Schaden Vernon
54637 Kohler Square #222, La Habra|(695)186-8469|Hessel Pasquale
45912 Dewayne Street, Mobile|090-709-3648 x282|Eichmann Garry
36649 Rippin Ports, Mentor|(069)989-8783 x644|Pfeffer Kelsie
30076 King Mews, Hayward|014-934-2377|Stiedemann Johnathon
78637 Florida Cliffs, Blythe|(234)229-3444|Casper Jayce
11608 Candace Court Suite 424, Cerritos|(822)060-1792|Becker Aurelie
45372 Penthouse, Jasper|(321)417-1788|Kub Heloise
49784 Schulist Ridge Suite 555, Temecula|583-988-4927 x940|Wolff Jaylin
23005 Morissette Fork Apt. 649, Florence|1-778-083-6571 x13579|Herzog Ally
85380 Robin Freeway, La Habra Heights|(776)957-0613|Lesch Ephraim
yaseen@yaseen-VirtualBox:~/CSC2001F/Practical_two/bin$
```

Comparison Experiment

Results



n	t
1	0,5565
500	0,6065
1000	0,8545
2500	1,16
5000	1,51
7500	1,922
10000	2,053
Average	1,2375



n	t
1	0,548
500	0,529
1000	0,511
2500	0,565
5000	0,530
7500	0,558
10000	0,550
Average	0,542

Discussion

Hypothetically the AVL Tree should be perform faster as it has O(logn) on its search operation whereas the Binary Search Tree only has O(n) in the worst case scenario.The graphs above display the correlation between the execution search time and the number of data entries in the electronic phonebook. The first graph displays the data using a Binary Search Tree (BST) data structure whereas the second uses an AVL Tree. The correlation results shown between the two variables (time and entries) are approximately 0.5 for the BST and 0.02 for the AVL search method. It is evident that times are much faster in searching for entries in the AVL data base. This is a direct result of a AVL cutting the range of values to search when the value inserted in the searchAVL function Is compared. The added conditions of the binary search tree used by the the searchAVL method a significant effect on the search time. Something that was unexpected was the fact that the execution times for the AVL even became smaller as more entries were given. And hence the code had to be done multiple times for an average time to be calculated. This yielded equivalent results and times were milliseconds apart. The first run of times were just adopted for comparison.

The searchAVL method in the worst case is of O(logn). This means it will act faster than traditional BST due to its added properties. The balancing of the tree cuts search time by half each time a level is descended in the data structure

Conclusion

Several conclusions can be drawn from this experiment. Firstly, the two search methods aren't close in terms of correlation coefficient even though both are Binary Search Trees. The AVL is as much as 40% faster, this was calculated from the average time of both methods for all values of n. The AVL also has a much better search functionality as it reduces the number of nodes to visit from the moment the search method is executed. The BST and the AVL both follow a logarithmic curve which implies that more entries result in better times. The ratio of time execution to number of entries given therefore decreases. To increase the results accuracy a better performing pc could have been used.