

CSC2001F Data Structures Assignment 2 Report – TSHBON035

Overall OOP design

In this Data Structures assignment, I was required to create an application (which contains the main method), called GenericsKbAVLAppGUI, which is implemented using an AVL Tree that uses a Binary Tree. There are 4 classes along with GenericsKbAVLAppGUI namely, BinaryTree (dependent on BinaryTreeNode), BinaryTreeNode, AVLTree (inherits Binary Tree) and the Generic class. Starting with the Generic class, it has 3 instances, which are, term, statement and cfScore, this class has a constructor, Getter (Accessor) and Setter (Mutator) methods, equals method (Overridden) and toString method (Overridden).

AVL Tree implementation (AVLTree)

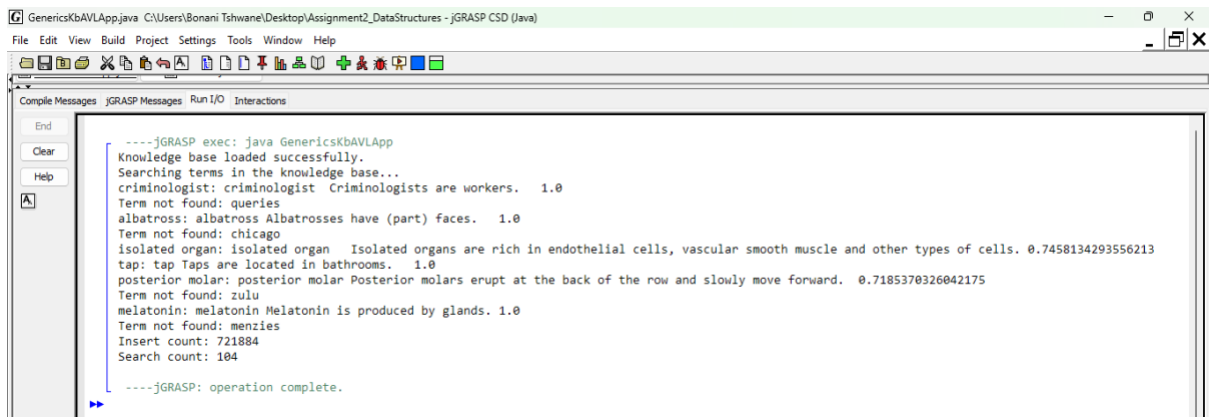
The AVLTree has 2 instances, opCountInsert & opCountSearch and the following methods; insert to load the file (with lines in Generic object format, term, statement, confidence score) into the knowledge base and add a new Generic object into the tree (Generic object being each line with format, term, statement, confidence score), find searches a Generic object in the tree recursively, based on the term provided from the queries file, height, balanceFactor, fixHeight, rotateRight, rotateLeft, balance, getOpCountInsert, getOpCountSearch. I made use of the AVL Tree from the lecture by Prof. Hussein Suleman where he referenced from kukuruku.co/post/avl-trees/ and modified it to suit how I wanted to use it in my application, so that it makes use of Generic class/type I created.

Experiment

Experiment description

The first part of this experiment is to first enhance the code to include instrumentation to count the number of comparison operations during insert and search operations. This involves adding variables to track the count and incrementing them wherever key comparisons occur. Then we want to analyse the performance of an algorithm by varying the dataset size (n) and measuring the number of comparison operations in best, worst, and average cases for insert and search operations. 10 values of n , logarithmically spaced (I used base 3 – 3^n), are chosen up to 50000. Randomized subsets of n entries are created for each value of n , and the instrumented application is run with fixed query files to track comparison operations. After collecting data for all values of n , we determine the minimum, maximum, and average count values for both insert and search operations. Finally, we use graphs to compare the experimental values obtained with the theoretical complexity analysis for insert and search operations, showing any similarities or deviations between them. This experiment provides insight into how the actual performance of the implemented AVL Tree aligns with its theoretical expectations across varying dataset sizes. I used a GUI called AVLGUI for the experiment which will then output the minimum, average and maximum cases.

Trial test values and outputs



```
----jGRASP exec: java GenericsKbAVLApp
Knowledge base loaded successfully.
Searching terms in the knowledge base...
criminologist: criminologist Criminologists are workers. 1.0
Term not found: queries
albatross: albatross Albatrosses have (part) faces. 1.0
Term not found: chicago
isolated organ: isolated organ Isolated organs are rich in endothelial cells, vascular smooth muscle and other types of cells. 0.7458134293556213
tap: tap Taps are located in bathrooms. 1.0
posterior molar: posterior molar Posterior molars erupt at the back of the row and slowly move forward. 0.7185370326042175
Term not found: zulu
melatonin: melatonin Melatonin is produced by glands. 1.0
Term not found: menzies
Insert count: 721884
Search count: 104
----jGRASP: operation complete.
```

Fig.1 Using “Tester-Queries.txt” file that I created to test 10 query values and the output in each case.

Results graphs (3ⁿ)

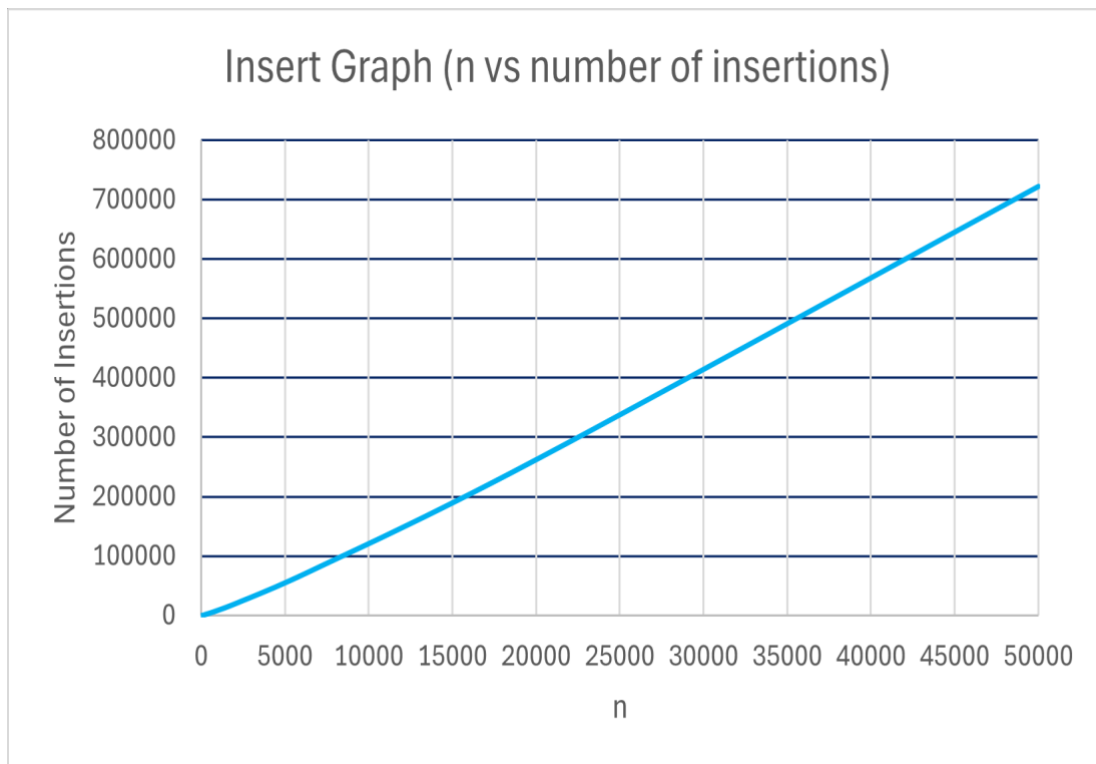


Fig.2.1 Complexity for insertion (best, average and worst cases).

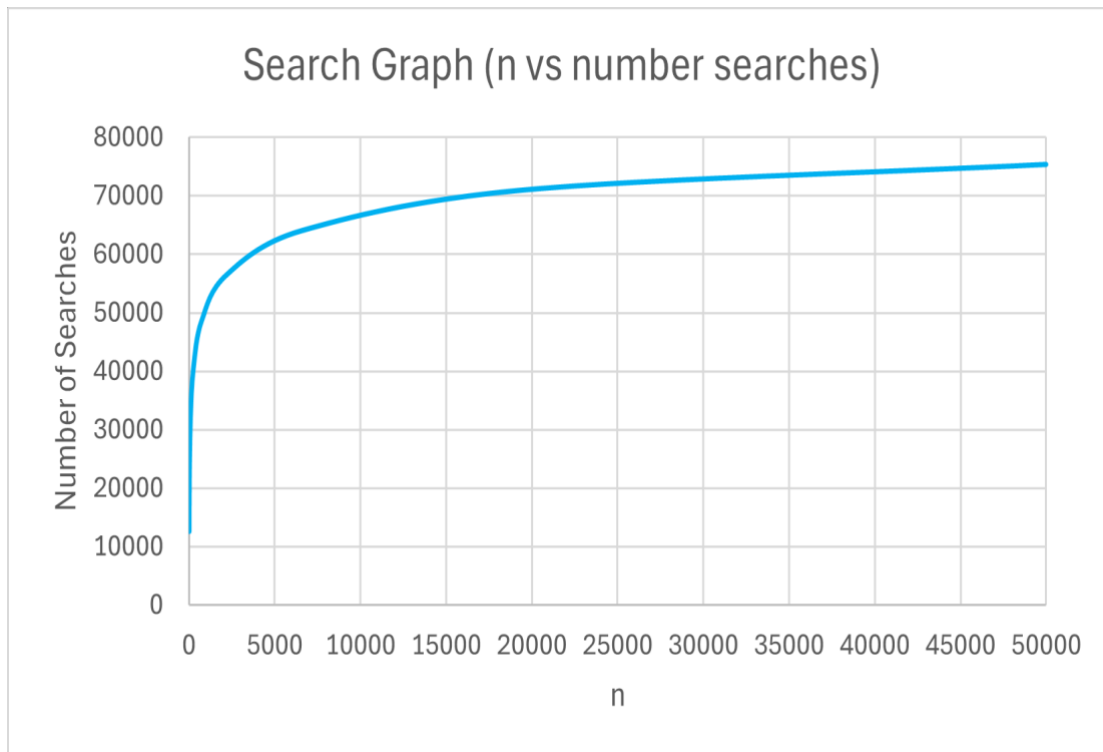


Fig.2.2 Complexity for searching best, average and worst cases).

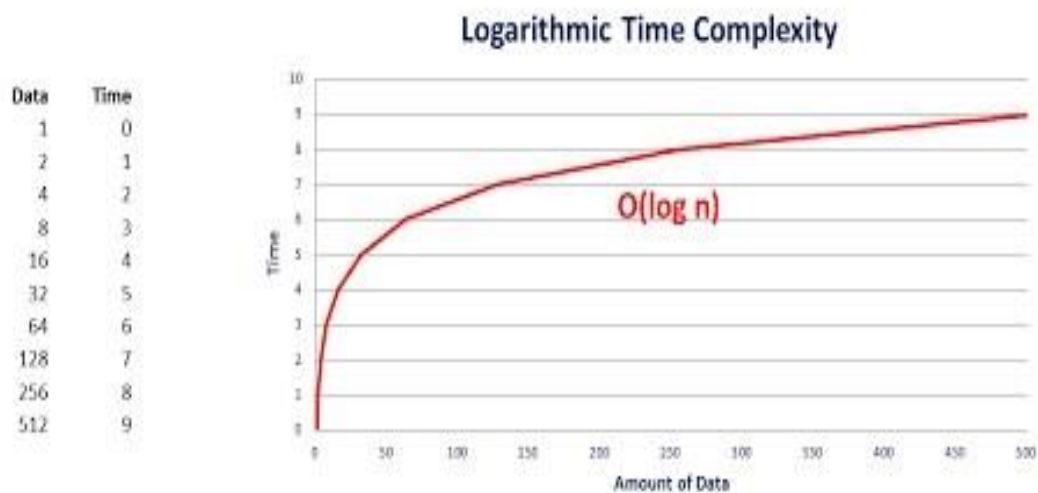


Fig.2.3 Theoretical (expected) time complexity for both insert and search operations (best, average and worst cases).

Discussion of results

The results presented in the graphs shown above illustrate the experimental performance of the AVL Tree's insert and search operations. I manually did the experiment by inputting the number of dataset size in each run so that I can vary, using base of 3 (3^n)S, the dataset 10 times using different n's. The experiment showed that AVL Trees perform constantly well on different dataset sizes. When measuring insert operations the graph resembled $n\log(n)$ and when measuring search operations it showed a logarithmic graph of $\log(n)$. This is consistent with the theoretical expectation of $O(\log n)$ for the best, average and worst case time complexity of the AVL Tree's insert and search operations. However, there are some variations between the experimental results and the theoretical expectation. When doing the experiment, I found and saw that the best, average and worst case for both the insert and search operations were similar which suggests that AVL Trees successfully balance themselves to ensure efficient performance in different/adverse situations, for example, different dataset sizes. These findings emphasize the advantage of AVL Trees when compared to regular Binary Search Trees. The experiment overall highlights the efficiency of an AVL Tree in when dealing with datasets of varying sizes.

Description of creativity

I created a GUI for GenericsKbAVLAppGUI that reads two files (GenericsKB.txt and GenericsKB-queries.txt) which then inserts GenericsKB.txt into the knowledge base and searches for the queries in GenericsKB-queries.txt in that knowledge base and returns the output. I created another GUI for the experiment part that counts for the maximum, minimum and average cases. I implemented this GUI using Java Swing.

Git log

```
0: commit 301eefe9d9e46baf9f072b0241acb0e27f72aba9
1: Author: Bonani Tshwane <tshbon035@nightmare.cs.uct.ac.za>
2: Date: Thu Mar 21 14:14:42 2024 +0000
3:
4: Created a GUI version of the GenericsKbAVLApp application which displays the graphs too.
5:
6: commit bb7afb64b7964e86cedf8d83e3ba00d407bf46bb
7: Author: Bonani Tshwane <tshbon035@nightmare.cs.uct.ac.za>
8: Date: Wed Mar 20 22:35:51 2024 +0000
9:
...
10: modified the app to take input that is n and will calculate the number of inserts and searches according to that number which gauges the GeberiksKB file in n's.
11:
12: commit 182fd7e83ffb9a6a7296dfb0ca0558ba9ca1d317
```

13: Author: Bonani Tshwane <tshbon035@nightmare.cs.uct.ac.za>

14: Date: Wed Mar 13 11:54:31 2024 +0000

15:

16: Created 5 java files that work together and the main method is on
GenericsKBAVLApp.java.