Report: Assignment 2 - AVL Data Structure

By Kelvin Wei (WXXKEL001)

A University of Cape Town Assignment Project

Table of Contents

OOP Design

The AVL Tree implementation code was re-used from the sample code provided. Minor adjustments were made to the AVL Tree to keep track of the number of comparisons. This included adding insert and search counters with getter methods. The sample AVL Tree inherits from the Binary Tree. A GenericData object stores data from the GenericsKB text file.

A Main class is created which performs the experiment and writes the experiment results to a text file. This data is in a directory called data.

Python was also used to help automate the graphing of data generated by the experiment. This includes two scripts (“plotting.py” and “ListGraphPoints.py”). “plotting.py” handles the reading in data and plotting of the data. “ListGraphPoints.py” stores the data from the text file in a way that makes it easier to plot the data.

Experiment and Testing

The goal of the experiment is to verify that AVL Trees are performance efficient with regards to their search and insert operations which follow an order of O(logN).

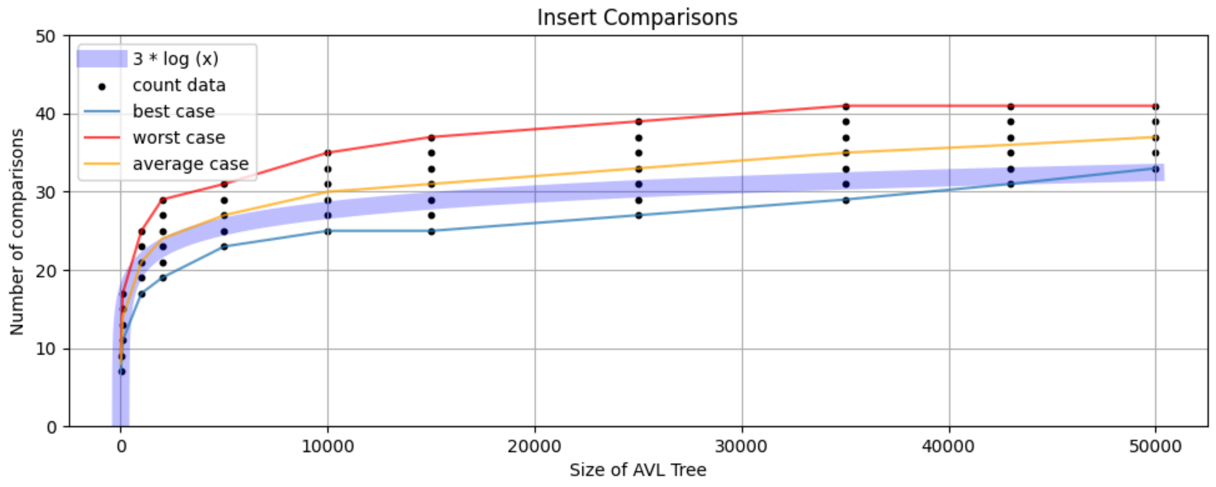
The experiment was performed by first loading a random subset of N entries from the GenericsKB.txt file. This subset of size N is then used to create an AVL tree of size N. The search queries from “GenericsKB-queries.txt” are then performed on the AVL Tree. An insert operation will be performed if the search query is not found and then deleted to preserve the size.

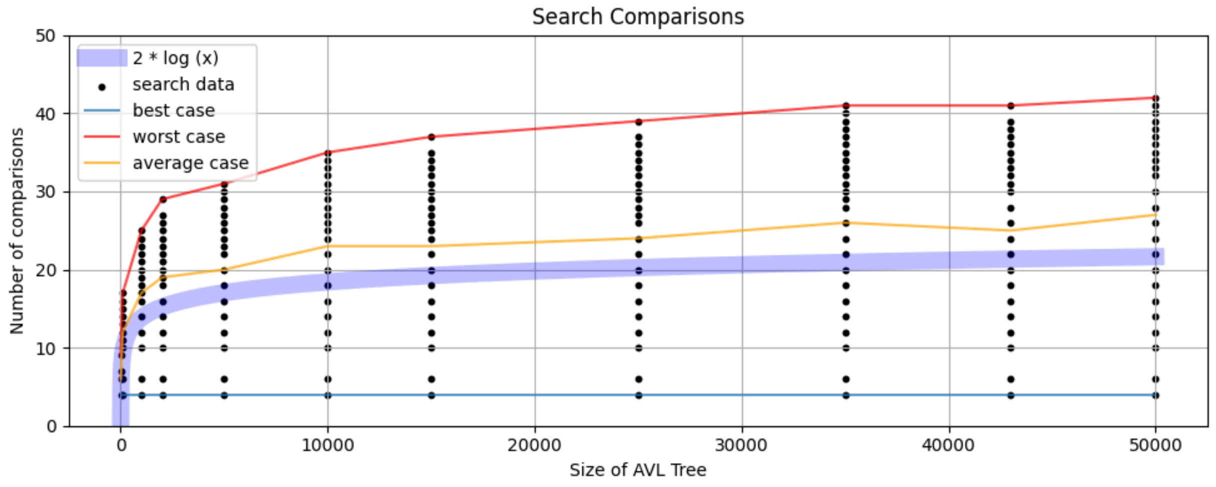
Each insert and search operation will have its count of comparisons(instrumentation) first stored in the program and then written to a text file. This data is then automatically graphed using python and displayed/saved.

Part 1: Test Queries and Output

```
Searching for: coastal city
Output: coastal city: Coastal cities are cities. (1.0)

Searching for: learn disability
Output: learn disability: Learn disability affects perceptions. (1.0)

Searching for: helpless kitten
Output: helpless kitten: Helpless kittens reach sexual maturity. (1.0)

Searching for: tellurium
Output: tellurium: Telluriums are minerals. (1.0)

Searching for: hemopoiesis
Output: Term not found: hemopoiesis

Searching for: zoning
Output: zoning: Zonings are division. (1.0)

Searching for: twentieth century
Output: twentieth century: A twentieth century is a century (1.0)

Searching for: conjoined twin
Output: conjoined twin: Conjoined twins are identical twins. (1.0)

Searching for: biological fitness
Output: biological fitness: Biological fitness is measured as number of
offspring. (0.7671203017234802)

Searching for: crimp
Output: Term not found: crimp
```

Part 2: Instrumentation Results





The data is plotted using a scatter plot which is represented by the black dots. The best, average and worst case is shown by the green, orange and red graph respectively. The thick blue transparent graph is an O(logN) graph which is used as a reference to the data. This is also shown in the respective graph's legend.

The most noticeable feature between the insert and search comparison graphs is how the range of values differs between the two. The search comparisons graph has a wider spread of values than the insert comparisons graph. This is because an insert operation would insert the new node as a leaf node and hence have to traverse the height of the tree. The search operation however can end on any level of the tree and doesn’t have to always traverse the height of the tree.
The best case for the search algorithm is when the root node is the desired node. This generates a straight line as seen in the search comparisons graph. The best case for insert is when the root is null. This isn’t present in the insert comparison’s graph since the tree is pre-built from a size of 10. Meaning the root is never null when the application is recording comparisons.

Creativity

During this assignment the whole running of the experiment, recording of data and graphing of the data is automated. This is achieved by using Java to run the experiment and record the data into a text file, python to graph and animate the data and the makefile to run the appropriate terminal commands to link everything together.

The makefile will compile and run the Java application experiment and create a virtual environment and install the necessary dependencies. It would then use the virtual environment for the python script to run on. The default makefile will run the animated graph. A static graph can be generated using `` `make plot` ``, `` `make plot_save` `` output the file as a png and `` `make plot_ani` `` to run the animated graph again.

In addition, in the Java program. A TreeMap and TreeSet are used to store unique data generated by the experiment since there is a lot of duplicate data. Storing the data in such a way would eliminate duplicate data and additionally store them.

Git Usage

```
git log | (ln=0; while read l; do echo $ln\: $l; ln=$((ln+1)); done) |
(head -10; echo ...; tail -10)
0: commit f52f750d1fa162e18d6918dfa2f46644de2f6502
1: Author: kelviy <52874397+kelviy@users.noreply.github.com>
2: Date:   Fri Mar 22 22:47:21 2024 +0200
3:
4: Added javadoc comments and generated javadoc into doc directory
5:
6: commit b3259b2f1c4798d987a00994ec92c76cd777de8b
7: Author: kelviy <52874397+kelviy@users.noreply.github.com>
8: Date:   Fri Mar 22 22:29:49 2024 +0200
9:
../..
61: Author: kelviy <52874397+kelviy@users.noreply.github.com>
62: Date:   Tue Mar 12 14:29:37 2024 +0200
63:
64: Added AVLTree code and plotting a log function
65:
66: commit b0bc4d722c20e278996894c6313fc67810c3acd9
67: Author: Kelvin Wei <wxxkel001@sl-dual-241.cs.uct.ac.za>
68: Date:   Tue Mar 12 10:26:47 2024 +0200
69:
70: Initial Commit of Git Repo
```