Assignment 2

CSC2001F

Chris Scheepers

SCHCHR077

22 March 2024

Object-Orientated Design

My object-oriented design is fairly rudimentary as I split up this task into five classes, namely:

- AVLTree.class
- AVLNode.class
- AVLExperiment.class
- AVLVisual.class
- GenericsKbAVLApp.class

I nested the “AVLNode” class within the “AVLTree” class because the “AVLNode” class serves no other purpose to the wider project and “AVLTree” is dependent on “AVLNode” so it does not necessarily need its own class. I created a separate class for “AVLTree” because it is a very useful piece of code that I can see myself using in later projects. Furthermore, it is good coding practice to modulate code with similar behaviours to make debugging, maintaining and understanding the code in the future easier as projects scale.

The “AVLExperiment” and “AVLVisual” classes are also their own classes for the above reasons. These classes are also not necessarily apart of the assignment’s requirements or specifications and so I felt there was no need to implement them within the “GenericsKbAVLApp” class. Separating these classes simply made it easier to understand my code as the project scaled as well as keep related methods and behaviours together, further helping keep the code tidy and the logic within each class understandable for people reading it for the first time.

The “AVLTree” class serves as the core representation of the AVL tree data structure, containing methods for insertion, rotation, searching, and utility functions. It collaborates closely with the “AVLNode” class, which defines the individual nodes within the AVL tree, each holding data and references to child nodes. These classes work together to construct and maintain balanced AVL trees efficiently. The “GenericsKbAVLApp” class acts as the main entry point, orchestrating interactions with the “AVLTree” class to load data, perform insertions, and execute searches based on user queries. Additionally, the “AVLExperiment” class conducts experiments on AVL trees, recording operation counts (search and insert) for different tree sizes. Finally, the “AVLVisual” class provides visualization capabilities by generating DOT representation files that are converted to a PNG for AVL trees created with the “AVLTree” class.

The Makefile has three run commands for each class with a main method:

run_avl: GenericsKbAVLApp run_exp: AVLExperiment run_visual: AVLVisual

Experimentation

The goal of the experiment was to determine the minimum (best case), maximum (worst case) and average number of operations that the constructed AVL tree would take for both search and insert functions. The theoretical efficiency of an AVL tree for both insert and search operations should be O(log(n)) that is the aim of the experiment – to determine whether the AVL tree I coded is indeed relatively close to that theoretical value.

Within "AVLExperimentation", I chose 10 values for *n* {3, 10, 33, 100, 333, 1000, 3333, 10000, 33333, 50000} that are equally spaced on a logarithmic scale for the sizes of the dataset. I then randomly populated lists of size *n* from "GenericsKB.txt" which were then used as datasets for the experiment. A for-loop of size ten (for each dataset) was used to then create a new AVL tree for each iteration of the for-loop. I built in search and insert counter methods within the "AVLTree" class that I then call to populate arrays insertOpCounts and searchOpCounts. The insert counter was incremented whenever a node == null, i.e.:

```
private AVLNode insertRec(AVLNode node, String data) {
        if (node == null) {
            insertOpCount++; // Increment insert operation count
            return new AVLNode(data);
        }
```

The search counter was incremented whenever a search operation was conducted, i.e. (<, =, >). Since the insert and search counters for each iteration were stored in arrays, it was easy to find the max, min and average values by using min(), average() and max() functions.

The results were then written into a .txt file and a .CSV file which I exported to Excel to then graph the data. I conducted this experiment 10 times to get more accurate results. Furthermore, I created the "AVLVisual" class that would read a .txt file and populate an AVL tree then create a .DOT file. With this .DOT file, it then converts the AVL tree into a .PNG so as to check visually that the AVL tree is balanced. As 50 000 elements is too large to display comfortably on an A4, I used datasets of 10 and 40 for this.

Test Values

The ten test queries I used to test “GenericsKbAVLApp” were as follows:

- soft drink
- maser
- onion ring
- concentration
- alcoholic cirrhosis
- nemertean worm
- diaphragm
- wild garlic
- medical receptionist
- competitor

The outputs for the test queries are as follows:

```
Enter the filename of the knowledge base: GenericsKB.txt
Enter the filename of the query file: queries.txt

Knowledge base loaded successfully.

Query: soft drink
soft drink: Soft drinks are beverages.

Query: maser
maser: A maser is an amplifier

Query: onion ring
onion ring: Onion rings are finger food.

Query: concentration
Term not found: concentration

Query: alcoholic cirrhosis
alcoholic cirrhosis: Alcoholic cirrhosis is the destruction of normal
liver tissue, leaving non-functioning scar tissue.
```

Query: nemertean worm

nemertean worm: Nemertean worms are long, thin, animals without segments.

Query: diaphragm

Term not found: diaphragm

Query: wild garlic

wild garlic: A wild garlic is a bulbous plant

Query: medical receptionist

Term not found: medical receptionist

Query: competitor

competitor: Competitors are located in sporting events.

Search Operations: 150

Insert Operations: 50000

Experiment Data

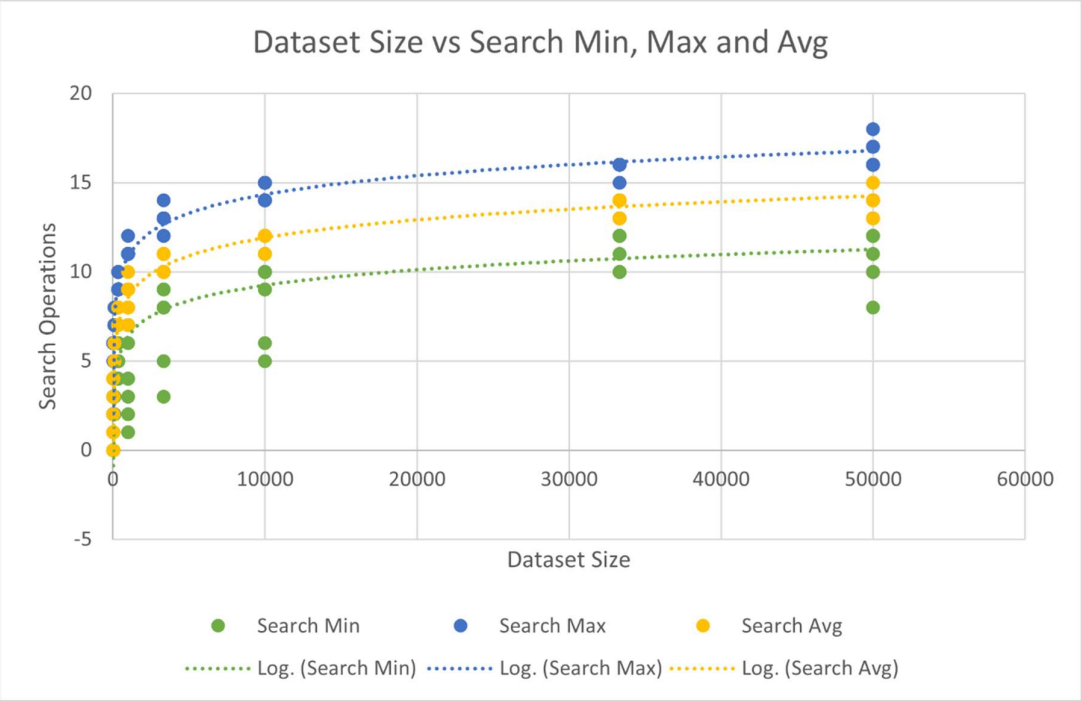

Fig 1.) Graph showing the spread of search operations vs dataset size

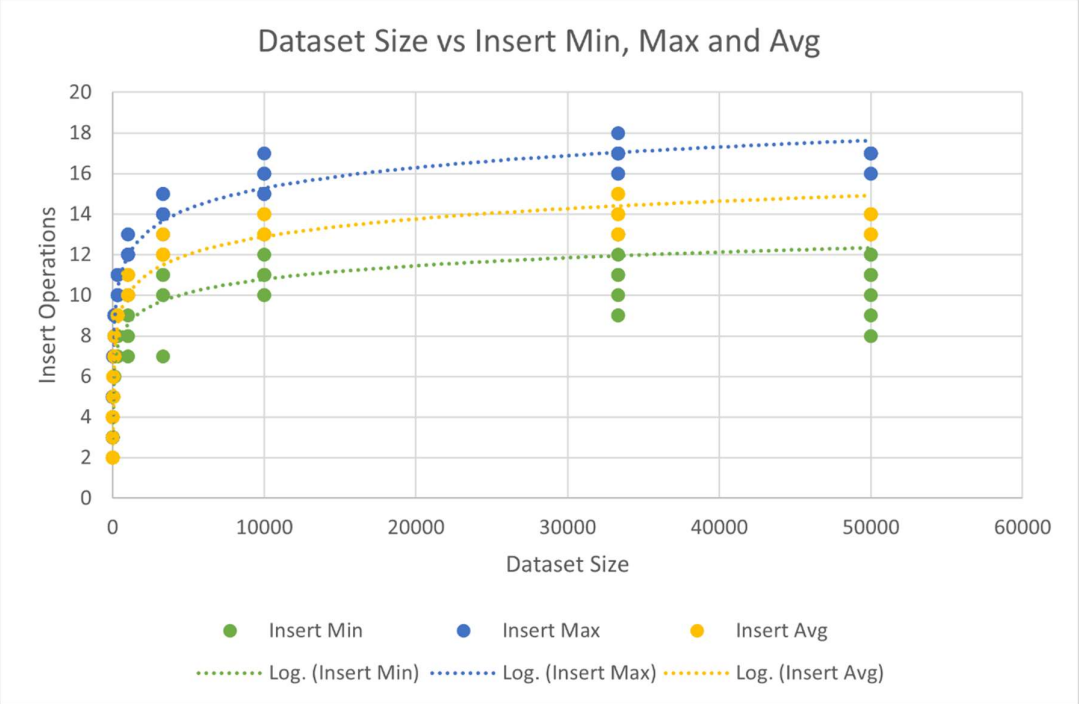

Fig 2.) Graph showing the spread of insert operations vs dataset size

Dataset Size	Insert Min	Insert Max	Insert Avg	Search Min	Search Max	Search Avg
3	2	3	2	0	2	0
10	4	5	4	0	3	2
33	5	7	6	1	6	3
100	7	9	7	3	7	5
333	9	11	9	5	9	7
1000	10	12	11	6	11	9
3333	12	13	12	9	12	10
10000	11	15	13	10	14	12
33333	11	16	14	10	16	13
50000	11	17	14	12	17	14

Fig 3.) Table of data from one experiment (out of 10) using the "AVLExperiment" class.

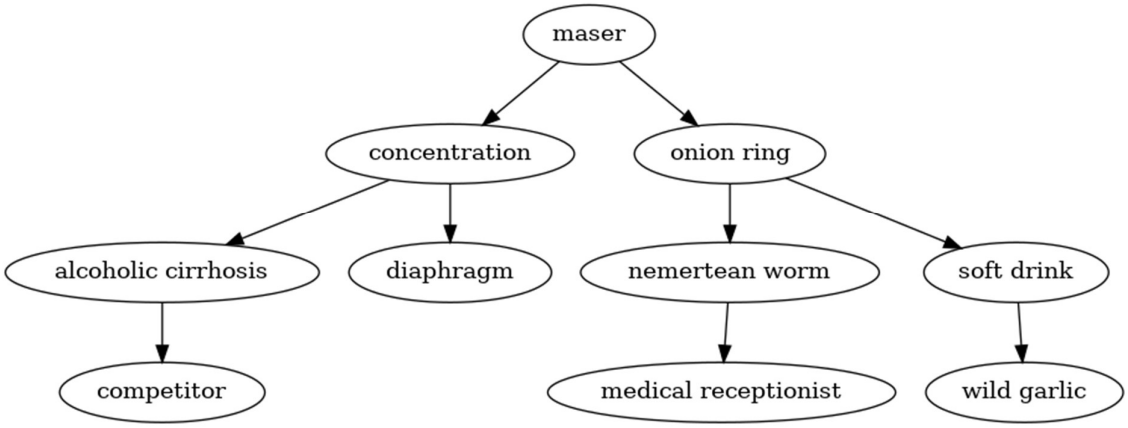

Fig 4.) Visual representation of a 10 element AVL tree using the class "AVLTree".

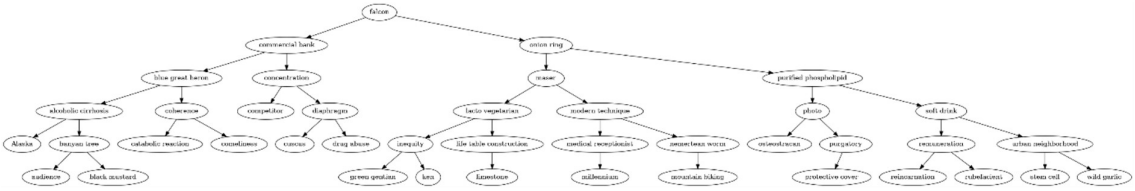

Fig 5.) Visual representation of a 40 element AVL tree using the class "AVLTree".

As shown in figure 2, it is clear that my implementation of an AVLTree is sound and works closely to the theoretical efficiency of O(log(N)) for both insert and search operations. As shown in figures 1 and 2, the data follows closely to a O(log(N)) efficiency, especially as the data sample increased. Furthermore, the data in table in figure 3 confirms the graphical representation by keeping roughly within the theoretical optimal performance. Figures 4 and 5 show that my implementation of the "AVLTree" class is correct and balances itself well.

Creativity

I think I implemented various functions within the assignment that went above and beyond the scope of this assignment. I believe implementation of the “AVLVisual” class, which is able to read from a text file and visually create an AVL tree using the “AVLTree” class, shows creativity as it benefits the debugging and experimentation process. I had to import a non-standard library (“graphviz”) but that shouldn’t be an issue for the marking process. The implementation of CSVWriter in my “AVLExperiment” class I believe also constitutes creativity as it greatly benefits the experimentation and recording of data processes. The method takes the dataset size, min, max and average of the search and insert values and exports them into a .CSV file that is then exported into Excel.

Git Log

First 10:

```
commit eeb1159db8fece44b427a443581c75caf1b4f369

Author: skippy <schchr077@myuct.ac.za>

Date:   Wed Mar 13 11:28:17 2024 +0200


    Beginning of file


commit 13e496b787e3aa1cb181551b1f6fa53ddb9f3be0

Author: skippy <schchr077@myuct.ac.za>

Date:   Wed Mar 13 11:41:10 2024 +0200


    Added a Makefile and text files into the project. Took the binary tree
from the last assignment and converted into an AVL tree.


commit 7003fc7da4b5bc961b4793a1d3c8a9a342d40ab7

Author: skippy <schchr077@myuct.ac.za>

Date:   Wed Mar 13 11:46:17 2024 +0200


    Added Javadocs to the source file.
```

```
commit 1bffd95aef4b797d1464462dd39a9da1df5c8eac
Author: skippy <schchr077@myuct.ac.za>
Date:   Wed Mar 13 11:47:47 2024 +0200

    generated the Javadocs.

commit ca52bae7a9a22ea0ba33bc5fe17fa38059743e09
Author: skippy <schchr077@myuct.ac.za>
Date:   Mon Mar 18 19:27:38 2024 +0200

    Fixed Makefile issues.

commit 741bb9820e694946511489233c713e3f6eb86d3b
Author: skippy <schchr077@myuct.ac.za>
Date:   Mon Mar 18 20:50:50 2024 +0200

    Added Query text file functionality. Removed terminal GUI from
assignment 1. Added a search and Insert counter.

commit 86d41383ad6f427f3136802b41a22e42efaa4704
Author: skippy <schchr077@myuct.ac.za>
Date:   Mon Mar 18 21:47:44 2024 +0200

    Fixing the search functionality (not finished)

commit 06eeb8622f803a33bf7708b9608ce0f001401629
Author: skippy <schchr077@myuct.ac.za>
Date:   Tue Mar 19 13:44:59 2024 +0200

    made an experiment Java file
```

commit f61bcd4b38e97aa0544daa60dd6fe08d92ec4e3e

Author: skippy <schchr077@myuct.ac.za>

Date: Tue Mar 19 13:46:31 2024 +0200

    updated Javdocs

commit d0bfdaf2cde2b51a48688874208018b3bea7a1d0

Author: skippy <schchr077@myuct.ac.za>

Date: Tue Mar 19 14:01:11 2024 +0200

    compartmentalized the AVLTree into its own Java file, fixed the
    Makefile.

Last 10:

Author: skippy <schchr077@myuct.ac.za>

Date: Fri Mar 22 16:22:06 2024 +0200

    Added some experiment results and added statement output when a term is found.

commit b81621589428ddba7a3a30907d3d2d0b463a8af1

Author: skippy <schchr077@myuct.ac.za>

Date: Fri Mar 22 16:30:30 2024 +0200

    added comments and javadoc documentation

commit 82c30d5c93bc71209936f17d56a72f0691e87fb6

Author: skippy <schchr077@myuct.ac.za>

Date: Fri Mar 22 16:34:50 2024 +0200

    added a git log

```
commit 17c3e439c4234390e95bfbfd1af815ada68ed08e
Author: skippy <schchr077@myuct.ac.za>
Date:   Fri Mar 22 16:38:17 2024 +0200

    generated docs

commit 4c3db5413795c8d0c4783de5d8ce5a5fc291b932
Author: skippy <schchr077@myuct.ac.za>
Date:   Fri Mar 22 18:01:33 2024 +0200

    changed dataset size

commit 3411671e8421acd5a69642a64f35b3f1d72f80ed
Author: skippy <schchr077@myuct.ac.za>
Date:   Fri Mar 22 19:06:32 2024 +0200

    Formatting.

commit 407f0cfacd52e4a83c7e6551e39b2557584afe27
Author: skippy <schchr077@myuct.ac.za>
Date:   Fri Mar 22 19:19:40 2024 +0200

    Fixed the insert counter.

commit b96f5634a7fa43fa875f17591467c23cd23b3b9e
Author: skippy <schchr077@myuct.ac.za>
Date:   Fri Mar 22 19:30:00 2024 +0200

    Fixed insertions counter and formatted document
```

```
commit 4ad378130518e6fecf044864f9f681522f3c1501
Author: skippy <schchr077@myuct.ac.za>
Date:   Fri Mar 22 19:45:07 2024 +0200

    //end of file
```