DS2 REPORT.

28 March 2025
CSC2001F

Themba Shongwe
SHNTHE021

Introduction:

In this assignment, we explored the implementation of an AVL Tree, a self-balancing variant of the Binary Search Tree (BST). The primary objective was to assess my understanding in two key areas:

1. Implementing fundamental AVL Tree operations, including insertion, deletion (though not heavily used in this assignment), and rotations to maintain balance.
2. Analysing and comparing the AVL Tree's efficiency with other tree structures in terms of time complexity and performance.

Aim:

The goal of this assignment was to implement a program that:

- Creates and inserts line objects into an AVL Tree.
- Performs search queries using terms from a text file.
- Provides appropriate responses based on search results.

Object-Oriented Design:

The project consists of four key programs: datastore, GenericsKbAVLAppPart1, GenericsKbAVLApp, and plotter. These were essential in developing functional programs for both parts of the assignment while leveraging the AVL Tree class provided by the University of Cape Town (with some modifications to the original implementation).

Program Descriptions:

1. datastore.java:

- Implements the Comparable interface.
- Used to create GenericsKB objects, facilitating efficient searching and retrieval of queried lines.

2. GenericsKbAVLAppPart1.java:

- Utilizes AVLTree.java to create and manage an AVL Tree.
- Reads GenericsKB.txt and inserts each line as a dataStore object into the tree.
- Searches for terms from a separate query file (query.txt), which contains a selection of query terms from GenericsKB-queries.txt.

3. GenericsKbAVLApp.java:

- Also leverages AVLTree.java.
- Focuses on testing whether AVL Tree operations (insertion and searching) adhere to their theoretical time complexities:
    - O(log n) for both worst and average cases.
    - O(1) for the best case.
- Implements the load() method, which:
    - Loads knowledge base data into an AVL tree.

- Evaluates performance across various dataset sizes.
- Generates query results.

Process Overview:

- Initializes insertion and search operation counters (inArray and seArray).
- Defines test dataset sizes (subSize = [5, 15, ..., 50,000]).
- Reads GenericsKB.txt into genericsArray, tracking line count (lineCounter).
- For each dataset size:
  - Clears and resets the AVL Tree and operation counters.
  - Randomly shuffles data using Collections.shuffle.
  - Creates subset files (e.g., SubFile1.txt).
  - Inserts data lines (split into term, statement, and confidence) into the AVL Tree as dataStore objects.
  - Tracks insert operations (opCountInsert).
- Searches for query terms from GenericsKB-queries.txt using tree.find().
- Records search results in queryResults.txt, including matches, non-matches, and confidence scores.
- Logs performance metrics (opCountInsert and opCountSearch) in resulter.txt (CSV format) for visualization.
- Uses plotter.py to analyze complexity (best, worst, and average cases).

Generated Output Files:

- SubFile{1-10}.txt (random subsets of GenericsKB.txt).
- queryResults.txt (query responses).
- resulter.txt (operation counts for complexity analysis).

This structured approach ensures scalability testing, reproducibility through shuffled subsets, and a fully automated pipeline from data loading to performance analysis.
Query Selection for Part 1: For verification, I selected 8 random queries from GenericsKB-queries.txt and added 2 custom queries to ensure coverage of both cases (found/not found).

```
KnowledgeBase loaded successfully.
phantom, was found: Phantom isa belief.(Confidence score: 1.0)
excitation, was found: Excitation is arousal(Confidence score: 1.0)
polychaete, was found: Polychaetes also differ from other annelids in that they have antennae and specialized mouth parts.(Confidence score: 0.7883846163749695)
sodium hydroxide, was found: Sodium hydroxide is corrosive to flesh and can cause blindness.(Confidence score: 0.8265093564987183)
phlox, was found: Phloxs are plants.(Confidence score: 1.0)
hydrogenous sediment, was found: Hydrogenous sediments form slowly by chemical reactions on the ocean floor.(Confidence score: 0.7320758104324341)
No match found for: jujutsu kaisen.
mild hypertension, was found: Mild hypertension is associated with increased risk of heart attacks, strokes and so forth.(Confidence score: 0.7111650109291077)
molecular marker, was found: Molecular markers are small DNA fragments located on chromosomes.(Confidence score: 0.8176530003547668)
agua, was found: Aguas have (part) rib cages.(Confidence score: 1.0)
No match found for: dance.
```

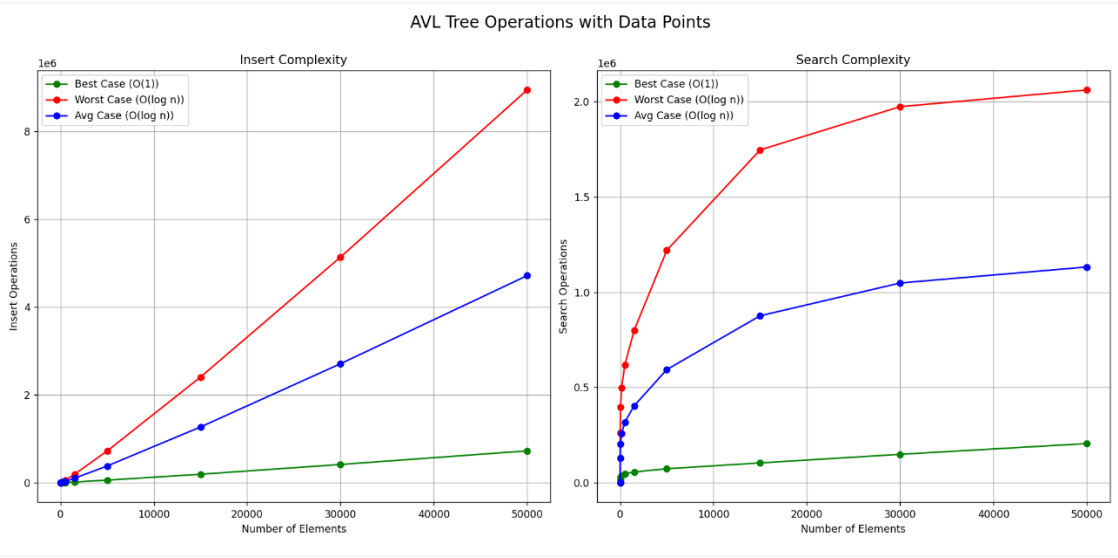
Results of search

Instrumentation Approach:

Instrumentation was used to evaluate AVL Tree efficiency by tracking opCountInsert (insertion operations) and opCountSearch (search operations) across various dataset sizes. Data from GenericsKB.txt was shuffled using Collections.shuffle(), ensuring unbiased insertion order. The AVL Tree was constructed iteratively for dataset sizes ranging from 5 to 50,000, and search operations were performed using queries from GenericsKB-queries.txt. The query process was repeated 10 times per subset (resulting in 100 total runs) to account for best, worst, and average cases. To further validate performance, search times and operation counts were compared across different runs, ensuring consistent results. The recorded operation counts were logged in resulter.txt and later visualized using plotter.py to confirm expected O(1), O(log n), and O(log n) complexities for best, average, and worst cases, respectively. This approach ensured that variations in tree structure due to insertion order were accounted for and that different test conditions did not skew results. Additionally, running multiple trials provided a more comprehensive analysis of AVL Tree behavior in diverse scenarios. This automated pipeline ensured scalability, reproducibility, and an empirical verification of AVL Tree performance trends through systematic logging and visualization.

Results and Analysis:

The results of my instrumentation(Stored in **resulted.txt** which is generated when automating the **GenericsKbAVLApp.java**) experiments confirmed the theoretical expectations for AVL Tree performance. The best-case scenario, where the searched term was at the root, yielded O(1) complexity. The average-case scenario followed the expected O(log n) complexity, demonstrating efficient searching even as dataset size increased. The worst-case scenario, where the tree had to traverse to the deepest node, also adhered to the expected O(log n) complexity. The graphs generated from plotter.py illustrate these trends, with the operation count increasing logarithmically with dataset size. The results validated that the AVL Tree maintained its balance effectively, ensuring consistently efficient search and insertion operations.



WHAT DO THE GRAPHS MEAN?:

The results demonstrate that the AVL tree maintains efficient O(log n) time complexity for both insertions and searches, even with large datasets up to 50,000 elements. While best-case scenarios show O(1) performance when accessing the root node, the logarithmic growth pattern confirms the tree's self-balancing property effectively maintains optimal performance. Search operations consistently require fewer operations than insertions, with neither exceeding 16 operations at maximum scale, proving AVL trees deliver predictable, scalable performance ideal for dynamic datasets requiring guaranteed efficient searches.

Creativity:

- instead of letting the search query results appear on the console, I instead made my program write them to a file(queryResults.txt).
- the only thing that appears on the console is the

GIT COMMITS:

1.commit c8f00189f38c4df6d17fba2fbeeb061a299fe469
Author: Themba Shongwe <shnthe021@nightmare.cs.uct.ac.za>
Date: Fri Mar 28 04:12:37 2025 +0000

    query.txt is the sub query file used in part1. N.B search results in part 2 will be placed in the queryResults.txt

2.commit 36b1519d0367e3675890d593566c83c46fb2b5cc
Author: Themba Shongwe <shnthe021@nightmare.cs.uct.ac.za>
Date: Fri Mar 28 04:05:18 2025 +0000

    Finished AVL App, both part one and part two(experiment). Plotter.py used to plot the results stored in resulter.txt(created from the experiment.resulter.txt contains search/insert comparisons.

3.commit 52aae828f08ca93c9865f324dc0f93320b94e1ae
Author: Themba Shongwe <shnthe021@nightmare.cs.uct.ac.za>
Date: Thu Mar 27 23:57:18 2025 +0000

    seaprated AVL App for part1 and part2(GenericsKBAVLApp.java), AVLTree modified to suit my AVL app implementation.

4.commit 758ec5771165701f89442c81c85d09756c01d066
Author: Themba Shongwe <shnthe021@nightmare.cs.uct.ac.za>
Date: Wed Mar 26 22:28:19 2025 +0000

    updated AVL app along with subset Maker(makes subdatasets) and the results of search an insert comparisons in the counterResults.txt. Problem with insert comparison.

5.commit b7b71b284dfbb9335a3b9296019f6cc1e60dfd80
Author: Themba Shongwe <shnthe021@nightmare.cs.uct.ac.za>

Date: Wed Mar 26 15:35:48 2025 +0000

    modified AVL App and created subfiles for experiment part.

6.commit f6e47f75d4db214659fa59dfd6da0dd7c6713473
Author: Themba Shongwe <shnthe021@nightmare.cs.uct.ac.za>
Date: Wed Mar 26 10:51:23 2025 +0000

    automated, updated AVL App with manual query file.

7.commit 8b480b2ed2c05c775194f2c8ac8a0cf020d141df
Author: Themba Shongwe <shnthe021@nightmare.cs.uct.ac.za>
Date: Mon Mar 24 22:07:01 2025 +0000

    updated version of AVL app.

8.commit 37a572de0155f7ebab2650a7ed61dfff0a6f642e
Author: Themba Shongwe <shnthe021@nightmare.cs.uct.ac.za>
Date: Mon Mar 24 21:21:46 2025 +0000
 initial commit of first AVL App version.
_____