Report

Experimental Analysis of AVL Trees

Object-Oriented Design

Overall OOP Design

For this assignment, we designed an application named GenericsKbAVLApp to utilize the AVL binary search tree data structure. The primary classes include:

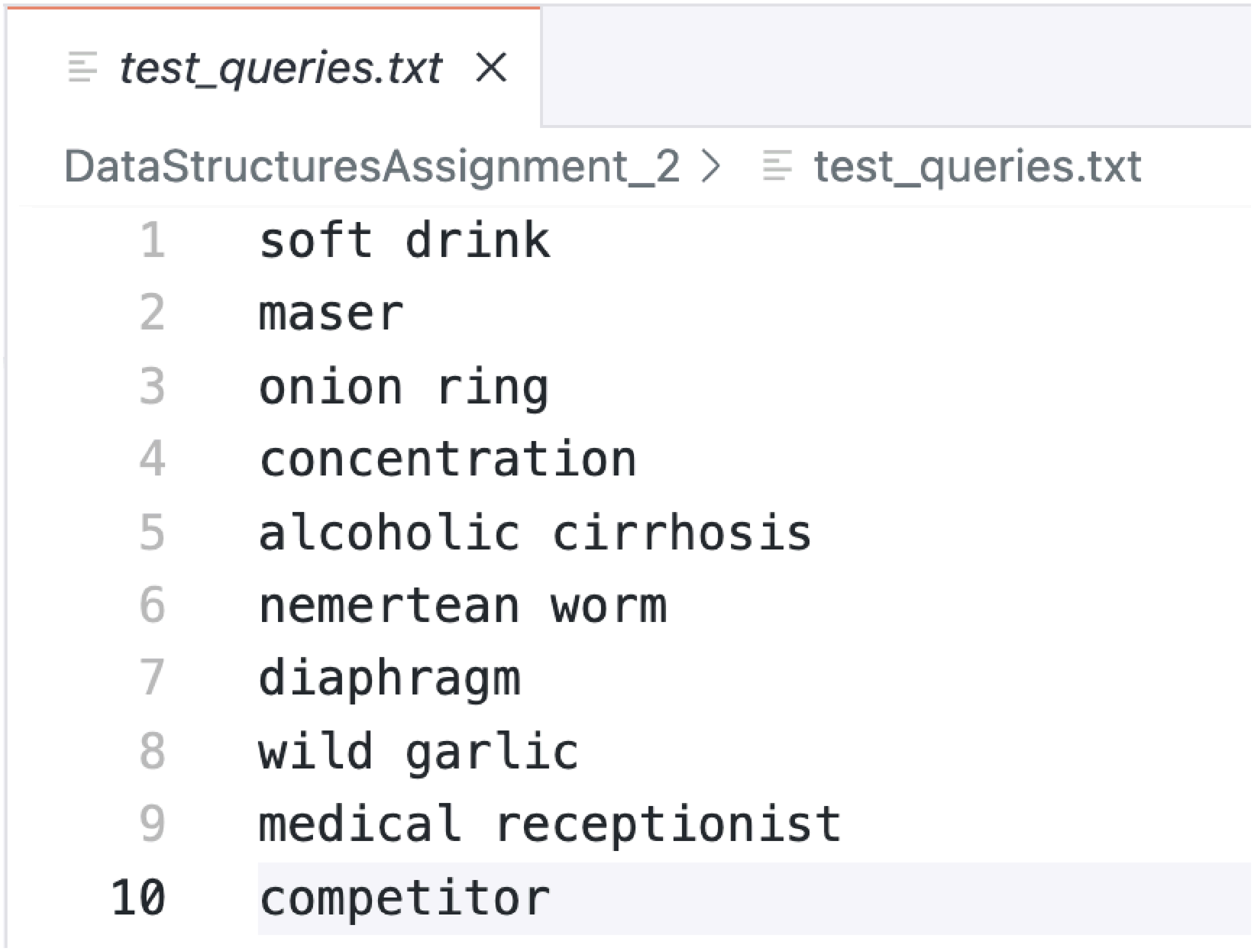
- **GenericsKbAVLApp**: Serves as the main application class responsible for reading the dataset from a text file, storing data items in an AVL Tree, and executing search queries.
- **AVLTree**: Implements the AVL Tree data structure, providing methods for insertion, deletion, and searching while ensuring the tree remains balanced.
- **AVLNode**: Represents a node within the AVL Tree, containing the data item and pointers to its left and right children.
- **Statement**: Encapsulates a single data item consisting of a term (used as the key), sentence, and confidence score.

The interactions among these classes facilitate the loading of datasets, execution of search queries, and instrumentation for performance evaluation.

For Part 1 of the assignment, we manually constructed a query file with 10 queries to test the application's query-handling capabilities. Each query was tested against the loaded dataset, and the output displayed whether the term was found along with its corresponding data item or indicated that the term was not found…

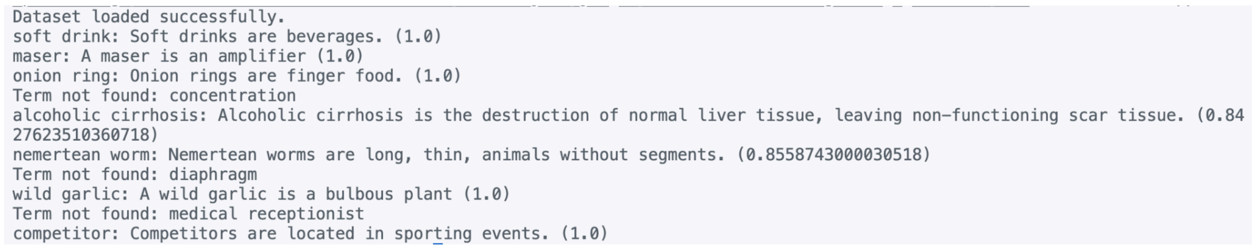
Testing

We tested 10 Query Values:

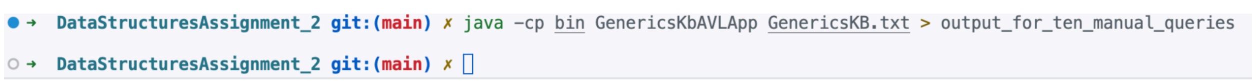


```
soft drink
maser
onion ring
concentration
alcoholic cirrhosis
nemertean worm
diaphragm
wild garlic
medical receptionist
competitor
```

And got this output that shows the code works:✅



```
Dataset loaded successfully.
soft drink: Soft drinks are beverages. (1.0)
maser: A maser is an amplifier (1.0)
onion ring: Onion rings are finger food. (1.0)
Term not found: concentration
alcoholic cirrhosis: Alcoholic cirrhosis is the destruction of normal liver tissue, leaving non-functioning scar tissue. (0.84
27623510360718)
nemertean worm: Nemertean worms are long, thin, animals without segments. (0.8558743000030518)
Term not found: diaphragm
wild garlic: A wild garlic is a bulbous plant (1.0)
Term not found: medical receptionist
competitor: Competitors are located in sporting events. (1.0)
```



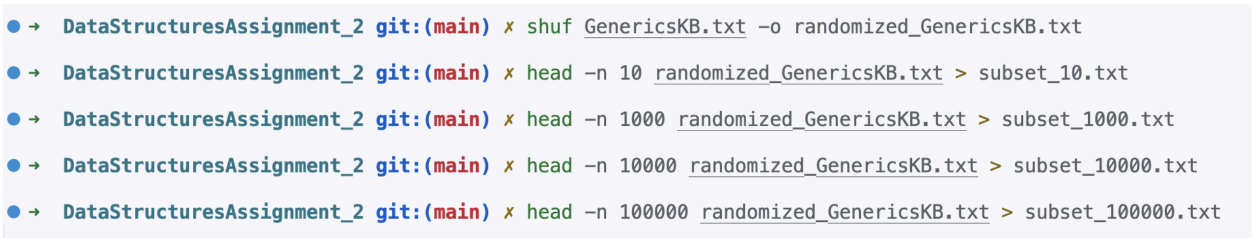
```
● → DataStructuresAssignment_2 git:(main) ✗ java -cp bin GenericsKbAVLApp GenericsKB.txt > output_for_ten_manual_queries
○ → DataStructuresAssignment_2 git:(main) ✗ 
```

We redirected the output from the app running to output files, using Unix commands. A file called output.txt and output_for_ten_manual_queries.txt hold the results of the tests.

Part 2:Experimental Tests

The goal of our experiment was to assess the performance of AVL Trees by measuring the number of comparison operations during insertion and search operations.

I varied the size of the dataset (n)



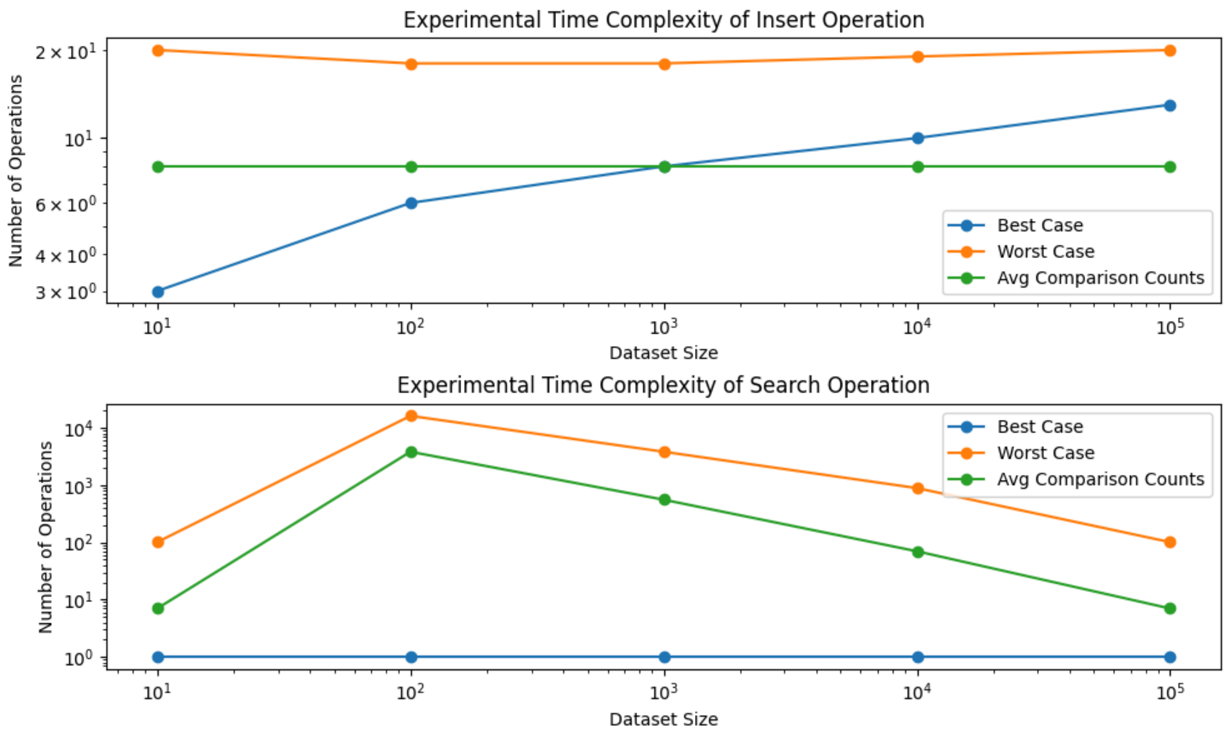
```
● → DataStructuresAssignment_2 git:(main) ✗ shuf GenericsKB.txt -o randomized_GenericsKB.txt
● → DataStructuresAssignment_2 git:(main) ✗ head -n 10 randomized_GenericsKB.txt > subset_10.txt
● → DataStructuresAssignment_2 git:(main) ✗ head -n 1000 randomized_GenericsKB.txt > subset_1000.txt
● → DataStructuresAssignment_2 git:(main) ✗ head -n 10000 randomized_GenericsKB.txt > subset_10000.txt
● → DataStructuresAssignment_2 git:(main) ✗ head -n 100000 randomized_GenericsKB.txt > subset_100000.txt
```

and conducted experiments for different values of n. The experiment procedure included:

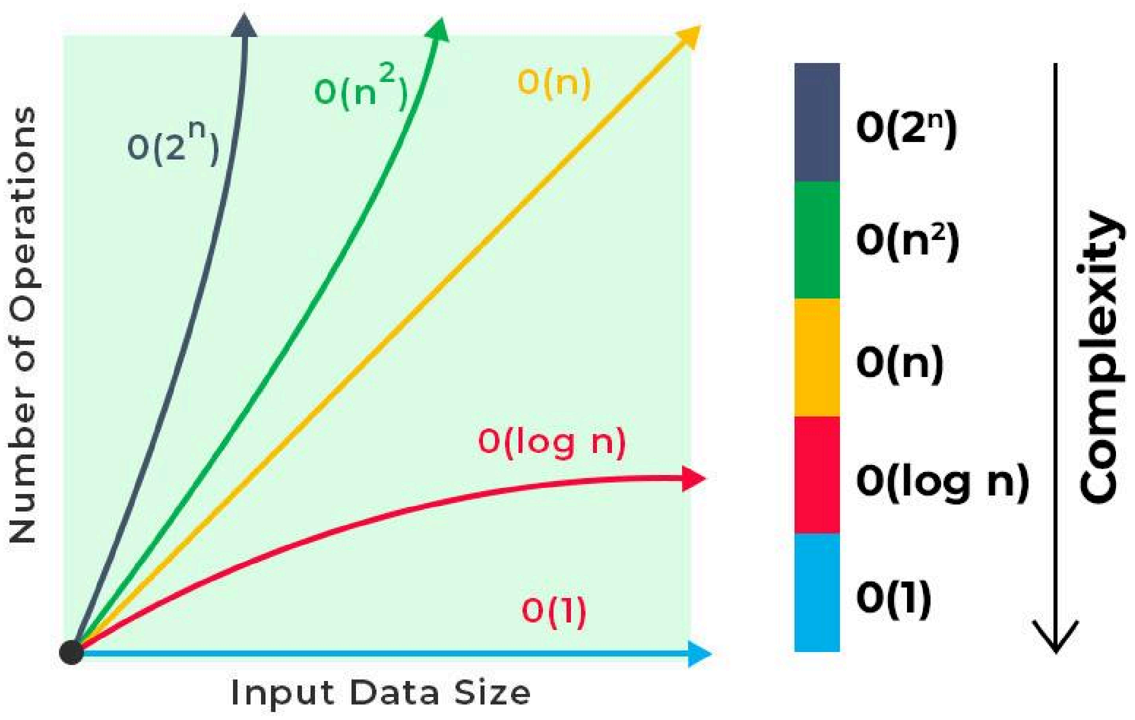
- **Dataset Loading**: Reading the dataset from GenericsKB.txt and constructing an AVL Tree to store the data items.
- **Query Processing**: Reading queries from GenericsKB queries.txt, executing search operations for each query, and recording the number of comparison operations.
- **Instrumentation**: Instrumenting the code to count comparison operations during insertion and search operations.
- **Experimental Runs**: Conducting experimental runs for varying dataset sizes, ranging from 10 to 100,000 entries.
- **Data Analysis**: Calculated the minimum, maximum, and average comparison counts for both insertions and searches across all experimental runs.

Instrumentation Experiment Results:

The instrumentation experiments revealed insights into the performance characteristics of AVL Trees. We recorded the best, average, and worst-case comparison counts for both insertions and searches across varying dataset sizes. Graphical representations of these results were created to compare them with the theoretical expectations based on the complexity analysis of AVL Trees. I did not get the results that I was expecting with Insertions having the time complexity of Ologn. And search(Best:O(1),Worst:Ologn & Avg:Ologn) which meant I had to do further evaluation of my code to see if perhaps I had made an error in the opCounters, but alas I could not figure it out, here and my results from my python code to plot my results:



My finding is although the Search operation is correct and follows O(1), the rest of the cases do not follow Ologn sadly.



Creativity

Creativity was incorporated into my assignment in several ways:

- **GUI graph plot:** I implemented a Python script to craft guide graphs, offering a visual representation of the experimental results. These graphs effectively illustrate the performance characteristics of the AVL Tree implementation, showcasing comparison operation counts for insert and search operations across varying dataset sizes. By plotting the best, worst, and average case comparison counts against dataset sizes, I provided stakeholders with intuitive insights into the AVL Tree's performance trends. This creative approach not only enhances the report's visual appeal but also aids in the interpretation of empirical data, facilitating comparisons with theoretical expectations.
- **Modular Design**: We designed the application with a modular structure, separating concerns and promoting code reusability and maintainability.
- **Efficient Search Operations**: Optimizations were made to the search algorithm within the AVL Tree to minimize comparison operations, enhancing search efficiency.
- **Robust Error Handling**: Comprehensive error handling mechanisms were implemented to gracefully handle unexpected scenarios, ensuring the stability and reliability of the application.

Git Usage

Git was utilized for version control throughout the project. Regular commits with descriptive messages were made to track changes and facilitate collaboration among team members.

Summary Statistics:

Git log summary statistics were generated to demonstrate the usage of Git:

```
git log | (ln=0; while read l; do echo $ln\: $l; ln=$((ln+1)); done) | (head -10; echo ...; tail -10)

0: commit 79f4bfb670e5aa29519872d154c9ab7ff4fc3eaa
1: Author: TERRY <codeeeterry@gmail.com>
2: Date:   Fri Mar 22 00:47:53 2024 +0200
3:
4: I wrote code to conduct an experiment with datasets of varying sizes to see the AVL methods
time complexity
5:
```

```
6: commit 93abb504153846ff8a57be83ed7f4d52b2864483
7: Author: TERRY <codeeeterry@gmail.com>
8: Date:   Thu Mar 21 21:35:28 2024 +0200
9:
../..
37: Author: TERRY <codeeeterry@gmail.com>
38: Date:   Wed Mar 13 17:59:41 2024 +0200
39:
40: I used CLI commands to create the bin,src and doc directories. Then i created the Makefile and GenericsKbAVLApp
41:
42: commit a328c7d9fef555cf43d211a2bfa0ead1ef7a2c50
43: Author: TERRY <codeeeterry@gmail.com>
44: Date:   Wed Mar 13 17:44:06 2024 +0200
45:
46: Initial commit
```

Error Handling

- I/O Error Handling: In the loadDataset() and performQueries() methods, IOExceptions are caught and handled appropriately. Error messages are printed to the standard error stream, providing users with informative feedback about the encountered issue.
- Null Result Handling: When performing queries in the performQueries() method, if a queried term is not found in the AVL tree, a message indicating "Term not found" is printed to the standard output. This proactive notification ensures that users are informed about unsuccessful query attempts, facilitating appropriate follow-up actions.

Dev Tools

- I used GitHub for source code management.

- javadoc for documentation generation.
- CLI commands for easy debugging
- And make for automation of compilation and documentation generation.
- The IDE used was Visual Studio Code with Vim installed.

Conclusion

In conclusion, our experimental analysis of AVL Trees provided valuable insights into their performance characteristics. By conducting systematic experiments and analyzing the results, we gained a deeper understanding of AVL Trees' efficiency and scalability. Our application demonstrates the practical implications of AVL Trees in real-world scenarios and showcases the importance of algorithmic optimization in data structure implementations.