GHRAHM004 – Ahmed Ghoor - CSC2001F Assignment 2

**Goal:**

Compare the efficiency of an AVL Tree and a Regular Binary Search Tree to insert and search through data.

**How:**

Created a regular Binary Search Tree App and an AVL Tree App to read in load shedding data from a file, insert them into the respective data structures and search through them for specific parameters.

Added counters to the code to record the number of comparison operations made when data is being inserted or searched. The count will be provided whenever the user requests an output that requires these operations.

Finally, in order to efficiently compare the data structures, additional applications were created to randomize the data in the input file, repeat the experiment multiple times to collect a large sum of useful data into convenient csv files and then summarize the data in those csv files so that understandable graphs may be plotted.

**Simple Objected Orientated class structure diagram**

**A screenshot of a cell phone

Description automatically generated**

**Part 2 & 4: Test values used in Trial runs with respective queried output & operations counts.**

1. Command: e.g. *java LSBSTApp “1\_2\_00”* OR *java LSAVLApp “blabla”*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Binary Search Tree (Regular) | | | AVL Tree | | |
| Values | Output | Insert Op’s | Search Op’s | Output | Insert Op’s | Search Op’s |
| 1\_2\_00 | 13 | 270100 | 3 | 13 | 34927 | 13 |
| 4\_22\_04 | 4, 12, 16, 8 | 270100 | 135 | 4, 12, 16, 8 | 34927 | 19 |
| 7\_22\_14 | 9, 1, 5, 13, 6, 14, 2 | 270100 | 309 | 9, 1, 5, 13, 6, 14, 2 | 34927 | 23 |
| 1\_2\_3 | Areas not found | 270100 | 30 | Areas not found | 34927 | 26 |
| 1 | Areas not found | 270100 | 6 | Areas not found | 34927 | 22 |
| blabla | Areas not found | 270100 | 408 | Areas not found | 34027 | 26 |

1. Command: (Output in Report is just the first and last 10 lines)
2. *java LSBSTApp*

1\_10\_00 15

1\_10\_02 16

1\_10\_04 1

1\_10\_06 2

1\_10\_08 3

1\_10\_10 4

1\_10\_12 5

1\_10\_14 6

1\_10\_16 7

1\_10\_18 8

…

8\_9\_08 7, 15, 3, 11, 8, 16, 4, 12

8\_9\_10 8, 16, 4, 12, 9, 1, 5, 13

8\_9\_12 9, 1, 5, 13, 10, 2, 6, 14

8\_9\_14 10, 2, 6, 14, 11, 3, 7, 15

8\_9\_16 11, 3, 7, 15, 12, 4, 8, 16

8\_9\_18 12, 4, 8, 16, 13, 5, 9, 1

8\_9\_20 13, 5, 9, 1, 14, 6, 10, 2

8\_9\_22 14, 6, 10, 2, 15, 7, 11, 3

Insert Operation Count: 270100

Search Operation Count: 0

1. *java LSAVLApp*

1\_10\_00 15

1\_10\_02 16

1\_10\_04 1

1\_10\_06 2

1\_10\_08 3

1\_10\_10 4

1\_10\_12 5

1\_10\_14 6

1\_10\_16 7

1\_10\_18 8

…

8\_9\_08 7, 15, 3, 11, 8, 16, 4, 12

8\_9\_10 8, 16, 4, 12, 9, 1, 5, 13

8\_9\_12 9, 1, 5, 13, 10, 2, 6, 14

8\_9\_14 10, 2, 6, 14, 11, 3, 7, 15

8\_9\_16 11, 3, 7, 15, 12, 4, 8, 16

8\_9\_18 12, 4, 8, 16, 13, 5, 9, 1

8\_9\_20 13, 5, 9, 1, 14, 6, 10, 2

8\_9\_22 14, 6, 10, 2, 15, 7, 11, 3

Insert Operation Count: 34927

Search Operation Count: 0

**Final Results for the comparison:**

1. Differences from above:

As briefly mentioned above, for my final comparison, I created an additional testing app to call the LSBSTApp and LSArrayApp repeatedly and outputted their results to a csv file in order to easily manipulate the data. Furthermore, I created an app to randomize the data in the input file in order to make the test results less predictable.

1. What the test does:

It calls the randomized file 10 times.

Each time, it calls a different number of lines in the file. Let’s call this n.

For each value of n, the testing program calls both Apps to insert and search for every value in that subset of n results.

(For example: if n=10, n lines of the file will be read, inserted into the different data structures, and then each of those 10 values will be searched for.)

Throughout this process, the number of operations for search and insertion are being calculated and I then send all this data is a csv file for ease of processing. [/bin/TestOutput/GraphData.csv]

I have summarized the data to provide the Worst, Best and Average Cases for each dataset and plotted them on graphs below:

**Summarised Data**

**Number of operations for search**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| DataSet length | BST Worst Case | AVL Worst Case | BST Best Case | AVL Best Case | BST AVG Case | AVL AVG  Case |
| 10 | 9 | 7 | 1 | 1 | 5 | 4 |
| 297 | 29 | 19 | 1 | 1 | 16 | 13 |
| 593 | 33 | 21 | 1 | 1 | 19 | 16 |
| 947 | 37 | 23 | 1 | 1 | 21 | 17 |
| 1243 | 37 | 23 | 1 | 1 | 22 | 18 |
| 1647 | 41 | 25 | 1 | 1 | 23 | 18 |
| 1943 | 41 | 25 | 1 | 1 | 24 | 19 |
| 2227 | 43 | 25 | 1 | 1 | 24 | 19 |
| 2543 | 43 | 27 | 1 | 1 | 25 | 20 |
| 2976 | 43 | 27 | 1 | 1 | 25 | 20 |

**Number of operations for insert**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| DataSet length | BST Worst Case | AVL Worst Case | BST Best Case | AVL Best Case | BST AVG  Case | AVL AVG  Case |
| 10 | 22 | 21 | 22 | 21 | 22 | 21 |
| 297 | 2348 | 2058 | 2348 | 2058 | 2348 | 2058 |
| 593 | 5552 | 4743 | 5552 | 4743 | 5552 | 4743 |
| 947 | 9654 | 8206 | 9654 | 8206 | 9654 | 8206 |
| 1243 | 13287 | 11245 | 13287 | 11245 | 13287 | 11245 |
| 1647 | 18599 | 15602 | 18599 | 15602 | 18599 | 15602 |
| 1943 | 22489 | 18890 | 22489 | 18890 | 22489 | 18890 |
| 2227 | 26286 | 22090 | 26286 | 22090 | 26286 | 22090 |
| 2543 | 30649 | 25721 | 30649 | 25721 | 30649 | 25721 |
| 2976 | 36809 | 30780 | 36809 | 30780 | 36809 | 30780 |

**Comments on Data:**

x-axis: Length of Dataset(n) y-axis: Number of comparison operators (search or insert depending on graph)

**Search Count:**

In terms of searching, the Binary Search Tree is much more efficient than the Array.

This is evident from the values on the y-axis. I actually had to separate the Data Structures onto 2 separate graphs because it was very difficult to see the shape of the lines for the Binary search tree when they were on one graph with the same y-axis.

**Insert Count:**

In terms of inserting values into the data structure**,** the Array is more efficient since there are no comparison operators needed. Hence, the comparison count is always zero.

The Average, Best and Worst case for the Binary Search Tree line was the same since the algorithm was inserting the same randomized file for all the tests. Therefore, the number of times that the insert comparison counter was called remained constant for each data set.

**Case for a bietjie creativity marks**

-Added a feature that allows you to call the program with a different data-textfile and/or dataset length.

-Created Extra Java apps (that can integrate with other java class should one wish to build on the current assignment spec) to test the data instead of running quick scripts in a foreign language.

-Check how I colour coded the tables to match the graphs. Even chose complementary colours. Man’s even creative in an artistic sense ekse.

**Git Log** (first and last 10 lines)