Assignment 2 Report

Michael Scott: SCTMIC015

OO Design:

LSItems:

* Object that takes two strings containing the “Stage, date and start time” of the load shedding and the areas affected by load shedding as parameters. Contains getter methods to obtain the information in the parameters. Contains a method that compares to Load Shedding objects and determines if they are equivalent.

BST:

* A modified Binary Search Tree class modified from [Java Program to Implement Binary Search Tree - Sanfoundry](https://www.sanfoundry.com/java-program-implement-binary-search-tree/).

BSTNode:

* It forms part of the Binary search tree implementation. Stores data from LSItems objects.

AVLTree:

* A modified Binary Search Tree class modified from [Java program to implement AVL Tree - Sanfoundry](https://www.sanfoundry.com/java-program-implement-avl-tree/)

AVLNode:

* It forms part of the AVLtree implementation. Stores data from LSItems objects.

LSBSTApp:

* Stored the Load shedding data objects in a binary search tree. Contained methods to search for a load shedding object given the “day, stage and start time”, a method to print out the object using an inorder traversal and a method to print results to a txt or csv file.

LSBSTAppTest:

* A modified version of the LSBSTApp that was used to write the both the insertCount and searchCount for insertions and search operations for each iteration through the rows of the data set. The data set could be input via the command line.

LSAVLAppTest:

* A modified version of the LSBSTApp that was used to write the both the insertCount and searchCount for insertions and search operations for each iteration through the rows of the data set. The data set could be input via the command line.

Experiment(Part 5):

The goal of the experiment was to demonstrate the speed difference in terms of operations count when performing a search between a binary search tree and an AVLTree. As well as to show the speed difference in terms of operations count when inserting LSItems into the two different trees.

Execution:

When performing the experiment, I carried out the following procedure:

1. Created 10 subsets of random data from the “clean” data file provided. The size of each subset was 297(i + 1) where i . This data was created using a python sript and was stored in 10 separate txt files.
2. Used a python script to pass the 10 txt files into both the “LSArrayAppTest” and “LSBSTAVLAppTest” programs. For each txt file passed as an argument the two programs would output a csv file containing each data item and the number of operations used when searching for it. This was done for both the search and insert operations
3. Used a python script (AutomationScript2.py) to collate the data from step 2 into two csv files containing the information from all the subsets for both the “LSArrayAppTest” and “LSBSTAppTest” for easy analysis.
4. The same python script would call upon another python script (graphs.py) which found the best, worst, and average case for all the sets of data.
5. The python script then compiled the data into graphs for search and insert operations for both data structures as well into two tables summarising the results for both the search and insert operations.

Results from part 1 – 4:

The results from parts 1 and 2 when implementing the LSBSTApp using values chosen by myself are as follows:

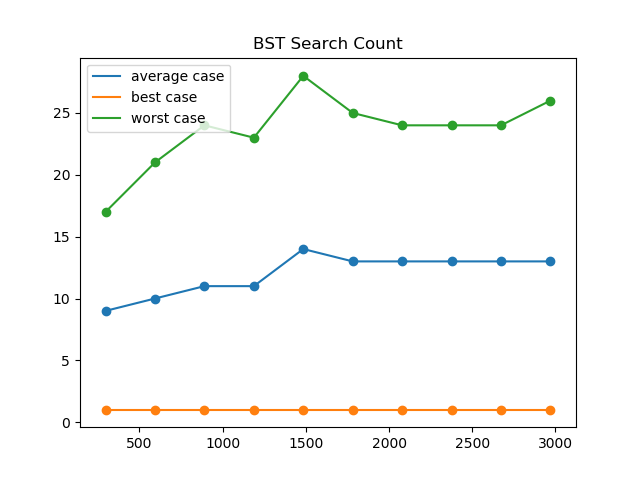
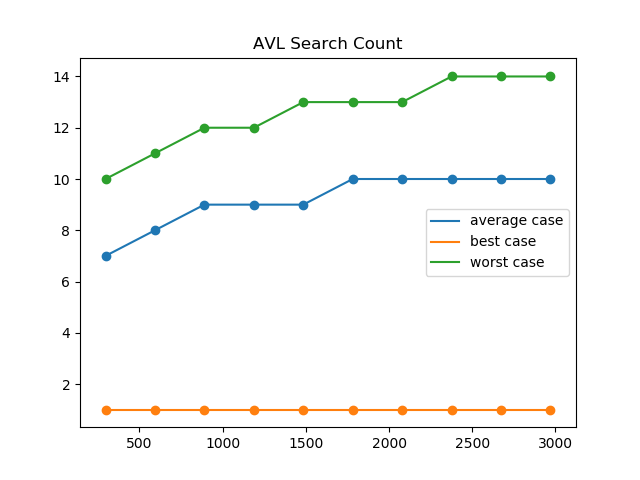
|  |  |  |
| --- | --- | --- |
| Parameter | Output | Operation count |
| 6\_15\_12 (Valid) | Data Structure: Array  Stage, date and start time tested: 6\_15\_12  Operations counted: 2155 | 2155 |
| 8\_21\_08 (Valid) | Data Structure: Array  Stage, date and start time tested: 8\_21\_08  Operations counted: 2678 | 2678 |
| 9\_14\_04 (Valid) | Data Structure: Array  Stage, date and start time tested: 2\_13\_04  Operations counted: 459 | 459 |
| 10\_11\_04 (invalid) | Areas Not Found | 2976 |
| 10\_18\_04 (invalid) | Areas Not Found | 2976 |
| No Parameter | Load Shedding Information: 1\_1\_00: Corresponding Areas: 1  Load Shedding Information: 1\_17\_00: Corresponding Areas: 1  Load Shedding Information: 1\_2\_00: Corresponding Areas: 13  Load Shedding Information: 1\_18\_00: Corresponding Areas: 13  etc.. |  |

The Results from parts 3 and 4 when implementing the LSAVLApp using values chosen by myself are as follows

|  |  |  |
| --- | --- | --- |
| Parameter | Output | Operation count |
| 2\_18\_10 (Valid) | Data Structure: Tree  Stage, date and start time tested: 2\_18\_10  Operations counted: 29 | 29 |
| 8\_18\_14 (Valid) | Data Structure: Tree  Stage, date and start time tested: 8\_18\_14  Operations counted: 184 | 184 |
| 6\_18\_18 (Valid) | Data Structure: Tree  Stage, date and start time tested: 6\_18\_18  Operations counted: 124 | 124 |
| 10\_11\_04 (invalid) | Areas Not Found | 126 |
| 10\_18\_04 (invalid) | Areas Not Found | 126 |
| No Parameter | Load Shedding Information: 1\_10\_00: Corresponding Areas: 15  Load Shedding Information: 1\_10\_02: Corresponding Areas: 16  Load Shedding Information: 1\_10\_04: Corresponding Areas: 1  etc |  |

Final Results from Part 5:

The Results for the experiment can be summarised as follows:

A close up of a building

Description automatically generatedTable 1) showing the results obtained for the search count for the LSBSTAppTest and LSAVLAppTest implementation with varied subset size (N)

We will first compare the search operations count between the two data structure using the graphs and tables presented above.

We can Cleary see from table 1 and figure 1 and 2 that both graphs have the same best case of a constant value of one. This occurs when we have a data item at the beginning of the array or at the root of the tree.

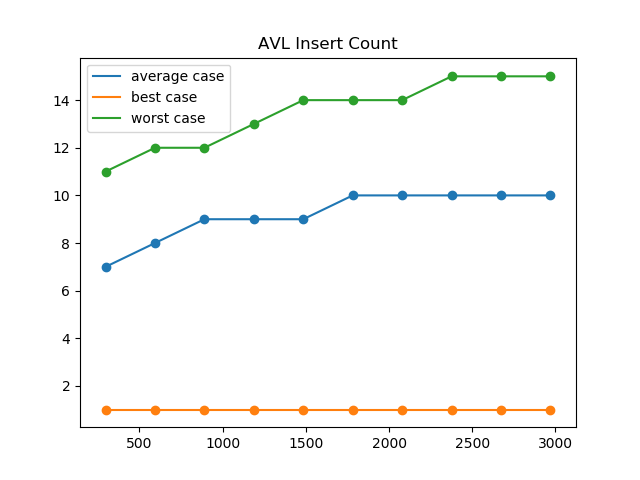
When comparing the average case we see a slight difference between the two data structures. The AVL Tree presents itself within the 7 to 10 range whilst the binary search tree presents itself within the 9 to 14 range.

However, we see a substantial difference when comparing the worst case for the two data structures. With the AVL Tree presenting itself within 10 to 14 range whilst the BST Tree presents itself within the 17 to 28 range. The BST requires nearly twice as many operations in the worst case.

We find the that both graphs take on a logarithmic trend for the search count. However, from looking at the graphs, we can see that the multiple of log(n) for the AVL is less than that for the BST Tree. Leading us to conclude that the greater the sample size becomes the more evident the difference between the two data structures is.

A screen shot of a social media post

Description automatically generatedTable 2) Showing the results obtained for the insert count for the LSBSTAppTest and LSAVLAppTest implementation with varied subset size (N)



We will now compare the insert operations count between the two data structure using the graphs and tables presented above.

As we can see from the figures and table the best case for both the AVL and BST Trees was 0. Which is where the tree was created.

The average case was very similar.

Creativity:

I went beyond the listed requirements in the project description in the following ways:

1. I created a python script to simulate the choosing of random subsets. This would allow for the most accurate results.
2. My python script also automated the process of loading the data into the data structures by passing arguments to the command line.
3. My script also created csv files for each data set and collated them at the end for ease access to the information.
4. I appended all my results from parts 2 and 4 to separate files in order to make it easier for me to track my results.

Git Log:

