UNIVERSITY OF CAPE TOWN
Department of Electrical Engineering

CSC2001 Assignment 2 Report

by Nicholas Antoniades ANTNIC008

March 22, 2019

1 | Questions

1.1 Aim of the experiment

The goal of this assignment is to compare the number of computations required to build and search through a typical balanced binary tree data structure to that of a AVL binary search tee. Both are to be implemented using Java and real power consumption data.

1.2 OO Design

PowerBSTApp

The PowerBSTApp class needed to read in a CSV file and create an array of objects for each line in the file. Then take that array, sort it, and then create a binary search tree with it. This array can then be searched through checking for matching instances. Five other classes were required when creating the PowerArrayApp class: timeStamp, textWrite, opCount, BinaryTree and Node.

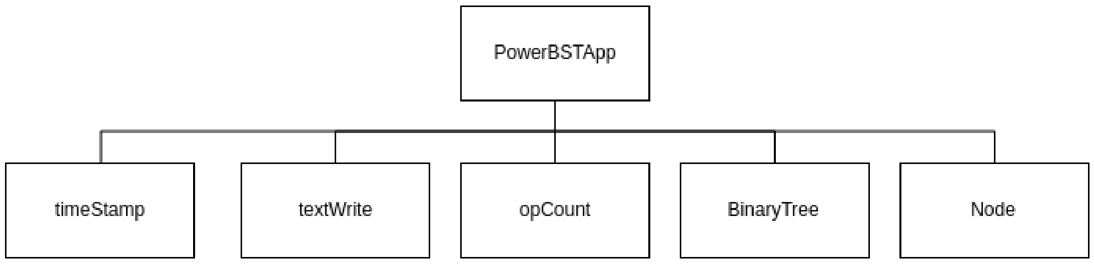

Figure 1.1: PowerBSTApp OO Design

PowerAVLApp

The PowerAVLApp class needed to read in a CSV file and create an AVL tree from the objects for each line in the file. This AVL tree can then be searched through checking for matching instances. Three other classes were required when creating the PowerArrayApp class: timeStamp, textWrite and opCount.

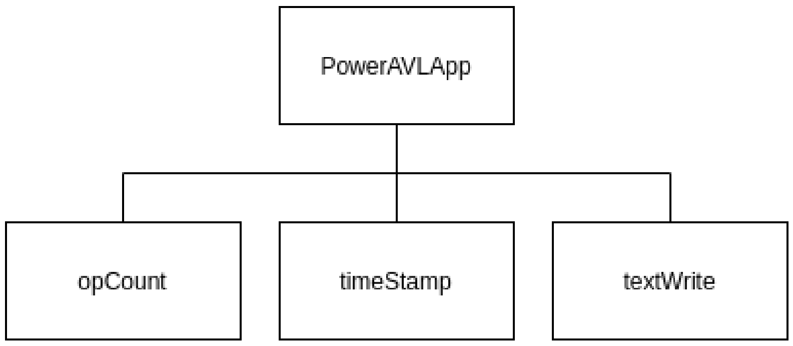

Figure 1.2: PowerArrayApp OO Design

Class descriptions

1. **timeStamp :** Used to read in a CSV and create an array of timeStamp objects which have the date, global active power and voltage.
2. **textWrite :** Takes in a timeStamp object and writes outputs it to a specified file.
3. **opCount :** Used to keep track of the total number of operations
4. **Node :** The object holding the date time information.
5. **BinaryTree :** Takes in an un-ordered array as an input and returns a balanced binary tree. This class allows for adding, searching and iterating through the binary tree.

Code references

The PowerBSTApp and PowerAVLApp applications include edited code implementations from William Fiset, william.alexandre.fiset@gmail.com, who put his his code up on GitHub for public use.

1.3 Testing

An initial test for both applications is done to check that they contain the correct data. The first 10 and last 10 lines of printAllDateTimes() for both PowerBSTApp and PowerAVLAPP apps for testing purposes can be seen below.

PowerBSTApp	**PowerAVLApp**
16/12/2006/19:51:00 233.220 3.388	16/12/2006/19:51:00 233.220 3.388
16/12/2006/23:20:00 241.580 1.222	16/12/2006/23:20:00 241.580 1.222
17/12/2006/00:29:00 243.680 0.612	17/12/2006/00:29:00 243.680 0.612
16/12/2006/20:35:00 233.370 3.226	16/12/2006/20:35:00 233.370 3.226
16/12/2006/17:37:00 232.910 5.268	16/12/2006/17:37:00 232.910 5.268
16/12/2006/19:23:00 234.360 3.334	16/12/2006/19:23:00 234.360 3.334
16/12/2006/18:25:00 233.740 4.870	16/12/2006/18:25:00 233.740 4.870
16/12/2006/20:30:00 234.540 3.262	16/12/2006/20:30:00 234.540 3.262
17/12/2006/00:32:00 241.860 2.376	17/12/2006/00:32:00 241.860 2.376
17/12/2006/00:22:00 240.560 0.276	17/12/2006/00:22:00 240.560 0.276
—	—
—	—
16/12/2006/23:15:00 242.390 0.386	16/12/2006/23:15:00 242.390 0.386
17/12/2006/00:42:00 243.650 0.382	17/12/2006/00:42:00 243.650 0.382
16/12/2006/23:52:00 238.890 3.458	16/12/2006/23:52:00 238.890 3.458
16/12/2006/18:38:00 234.020 2.912	16/12/2006/18:38:00 234.020 2.912
16/12/2006/17:41:00 237.060 3.430	16/12/2006/17:41:00 237.060 3.430
16/12/2006/19:21:00 234.020 3.332	16/12/2006/19:21:00 234.020 3.332
16/12/2006/23:47:00 241.230 2.540	16/12/2006/23:47:00 241.230 2.540
17/12/2006/00:09:00 242.090 0.838	17/12/2006/00:09:00 242.090 0.838
16/12/2006/22:01:00 237.680 1.786	16/12/2006/22:01:00 237.680 1.786
16/12/2006/17:43:00 235.840 3.728	16/12/2006/17:43:00 235.840 3.728

Table 1.1: Output values

Implementing the operation counters

By adding additional code to the BST and AVL methods, the number of comparisons could be discretely counted. All operations (<, >, =) that are being performing in the code are counted for both the method building the tree as well as for the method used to search through the treed. At the end of a search/addition of a node in the array the total operation count for both cases is then written to an output text file. Each test case will output to a separate file.

Running the tests using a bash script

A bash script was created in which both applications were tested using multiple test cases for a range of subsets of the given power data. Three data structures were tested. An AVl tree built from a unsorted data set, an AVL tree built from a sorted data set, and a balanced BST which is built from a sorted data set. The trees were tested using command line inputs to adjust data set size, specify an output textfile and read in a text file of keys to be searched for within the built data structure. The data set sizes ranged from 1 to 501 elements of data increasing in increments of 20, running each test case at every increment.

In order to fully test the implementations, in each set of keys there are two keys which have invalid keys. Testing response invalid symbols, non existant data, add a blank key. The test values that were used in the test for Part 2 and Part 4 used 3 sets of keys, these sets of keys were randomly taken from the overall data set in sets of 20. The keys can be seen in the table below.

Key set 1	**Key set 2**	**Key set 3**
16/12/2006/19:51:00	16/12/2006/19:40:00	16/12/2006/18:38:00
16/12/2006/23:20:00	16/12/2006/23:10:00	16/12/2006/19:05:00
qwffwf!!	16/12/2006/17:48:00	16/12/2006/19:51:00
16/12/2006/23:25:00	16/12/2006/17:36:00	
16/12/2006/23:45:00	17/12/2006/00:29:00	16/12/2006/20:00:00
16/12/2006/22:20:00	16/12/2006/20:32:00	17/12/2006/01:35:00
16/12/2006/22:18:00	17/12/2006/00:48:00	16/12/2006/20:47:00
16/12/2006/23:16:00	22:43:00	17/12/2006/00:32:00
16/12/2006/17:44:00	16/12/2006/22:38:00	16/12/2006/23:12:00
16/12/2006/21:05:00	16/12/2006/18:02:00	16/12/2006/21:27:00
16/12/2006/22:56:00	16/12/2006/21:51:00	17/12/2006/00:09:00
16/12/2006/17:30:00	16/12/2006/21:52:00	17/12/2006/00:32:00
991177	16/12/2006/23:39:00	*****///qwqw
16/12/2006/23:14:00	17/12/2006/01:18:00	16/12/2006/20:48:00
16/12/2006/21:25:00	16/12/2006/21:29:00	17/12/2006/00:09:00
17/12/2006/01:07:00	16/12/2006/20:53:00	16/12/2006/20:20:00
17/12/2006/01:28:00	16/12/2006/23:09:00	16/12/2006/20:20:00
17/12/2006/00:09:00	16/12/2006/	16/12/2006/18:09:00
16/12/2006/22:01:00	16/12/2006/21:53:00	16/12/2006/20:43:00
16/12/2006/17:43:00	16/12/2006/20:30:00	16/12/2006/21:13:00

Table 1.2: Test values

1.4 Results

1.4.1 Seach Operation count

As expected from a binary search tree it can be seen for the AVL and binary trees the number of operations required for each search was $log(n)$.

PowerAVLApp

For the PowerAVLApp two sets of tests were done. One where the array the tree was built from was ordered, and the other array was unorderd.

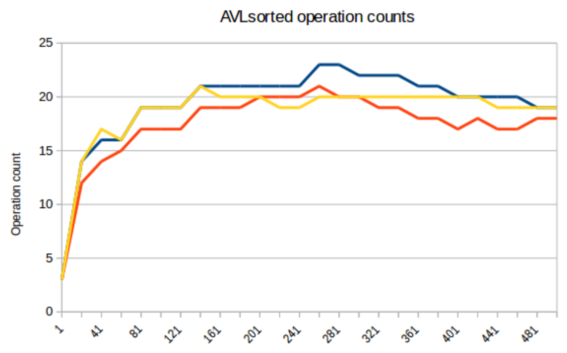

Figure 1.3: PowerArrayApp test results

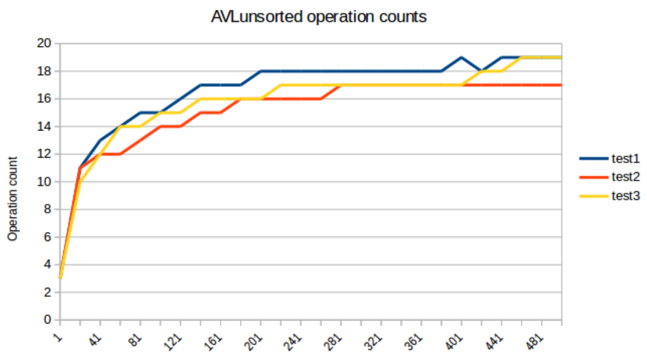

Figure 1.4: PowerArrayApp test results

It can be seen from the graphs above that an AVL tree built from unsorted data results in a tree structure that requires more operations to do a search than to that of an AVL tree built from unsorted data. The unsorted data results in a best case of 16 operations average of 17, worst case 19 and the sorted data results in a best case of 16 and average of 22 operations.

PowerBSTApp

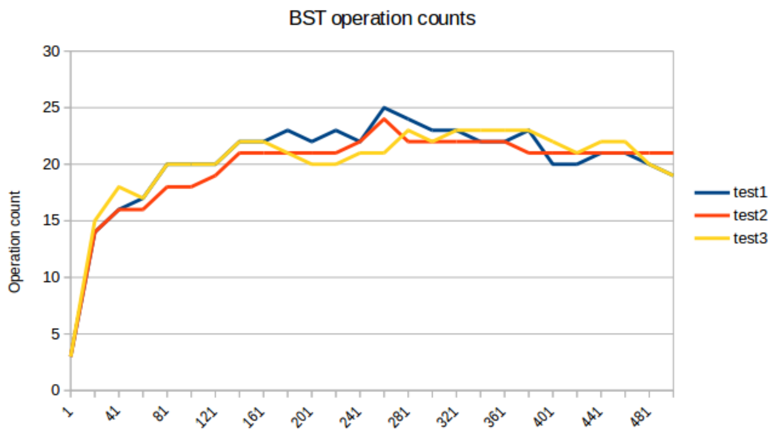

Figure 1.5: PowerBSTApp test results

The BST resulted in a best case of 20 operations, average of 22 and worst case of 25 operations.

Average search count operation

Once the average of the operation counts for the search through the 3 sets of keys was calculated, the results were then plotted on graphs using excel.

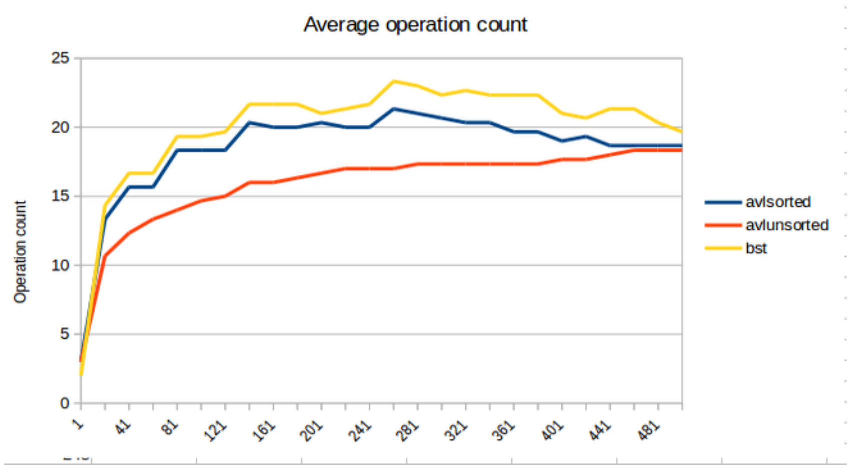

Figure 1.6: PowerArrayApp test results

It can be seen that the AVL tree built from unsorted data on average required fewer operations per search than that of the other two trees. The search in the AVL tree for the unsorted data required more computations than the one from the unsorted data but took less computations than that of the BST.

1.4.2 Build Operation count

For the each set of test keys the number of operations required to build the data set would be the same for the same size data set N because the same data set was being used. It can be seen that as N increases so does the number of operations required to build the tree which is expected. It can be seen that the AVL tree takes more operations to build the tree than the standard BST tree but this can be expected with the extra operations used when balancing the with the addition of each new node.

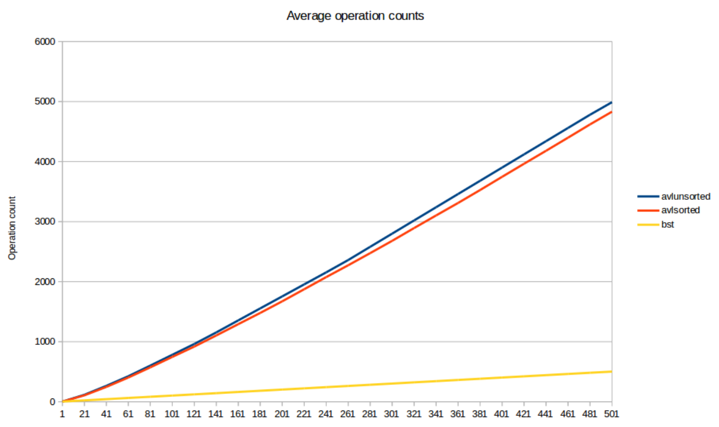

Figure 1.7: PowerBSTApp test results

1.5 Review and Summary

The PowerAVLApp and PowerBSTApp applications allow the user to search through a list of data for the power information for a specific time stamp. Using different input arguments, a set of keys to be searched can be specified, as well whether or not to display that operation count for a specific search. Other arguments implemented for testing purposes allows a specified data set size, and an output text file for the operation counts to be written to.

By running the bash script file the test results used in the report can be reproduced. All arguments required to run the programme can be found in the ReadMe file.

git log

```
1: commit 4311d381855f2f0cfbf6264663f19280b9ef1c7c
2: Author: Nicholas <antnic008@myuct.ac.za>
3: Date: Sun Mar 17 01:12:21 2019 +0200
4:
5: Created three test cases
6:
7: commit 757ba0b8614e13ad5b6a7e596e3b093843ca8f3b
8: Author: Nicholas <antnic008@myuct.ac.za>
9: Date: Sun Mar 17 00:25:32 2019 +0200
10:
-10: command not found
...
212: Author: Nicholas <antnic008@myuct.ac.za>
213: Date: Sat Feb 23 12:16:04 2019 +0200
214:
215: Initial files
216:
217: commit f25048e125ed9b497346ba407e5f8c0127903155
218: Author: Nicholas <antnic008@myuct.ac.za>
219: Date: Fri Feb 22 19:02:45 2019 +0200
220:
221: This is my first attempt
```