CSC2001F Assignment 1

Edson Shivuri SHVNKA005

Class Structure, Interaction and OOP design:

Array Classes:

LSArrayApp – Class containing the methods to implement the array solution. This class contains a main method from which to run the array solution and is also responsible for generating output.

BST Classes:

BTNode – Node class for a binary tree that stores a ScheduleItem object as well as a reference to the child nodes.

BT – Super Class for Binary trees. It therefore contains methods to be used by different types of binary trees. In this case, the BST class inherits from it.

BST – Class that extends the BT class and includes find and insert methods that recursively perform their tasks of respectively adding new data to the BST or searching for a specific element

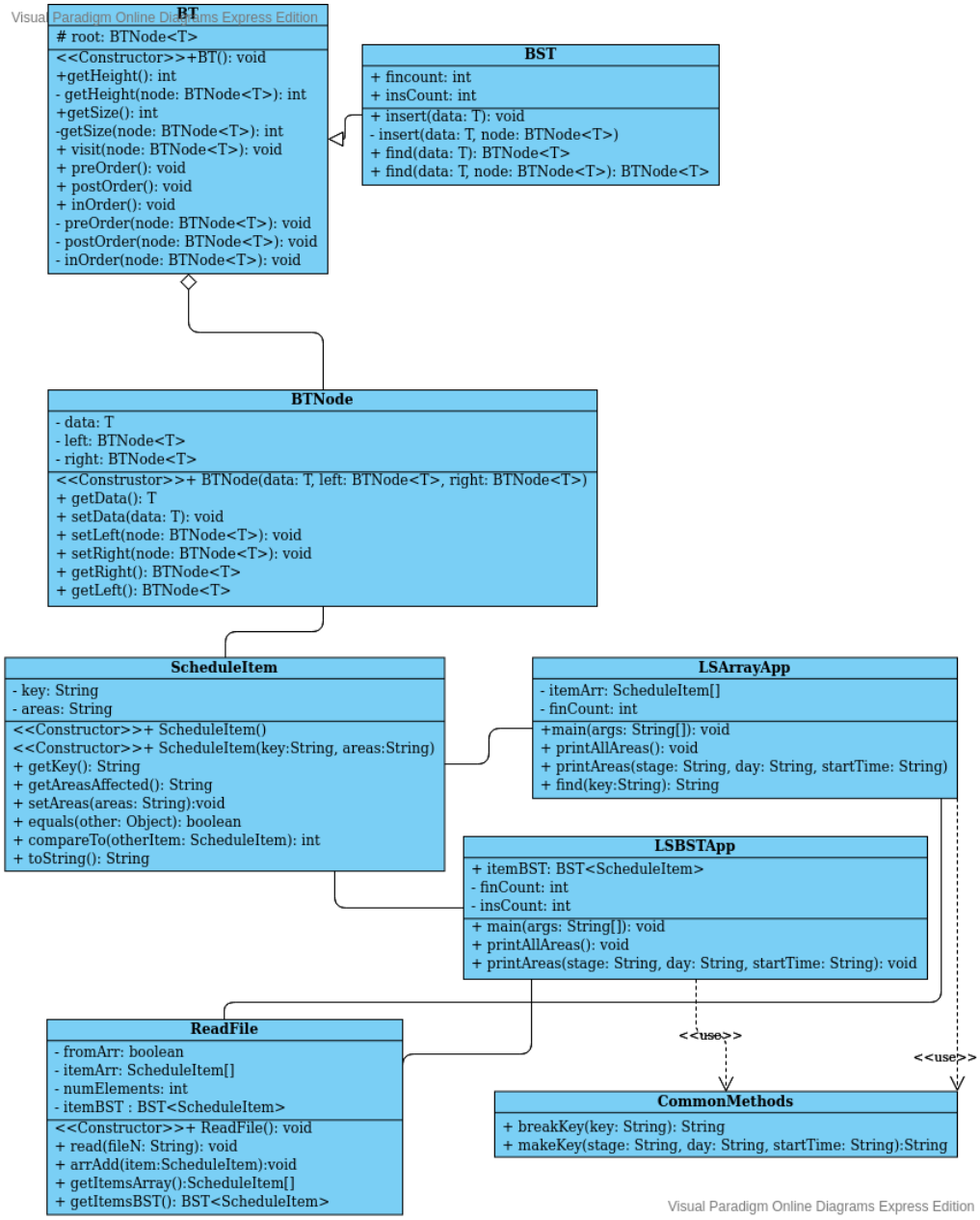
LSBSTApp – Class containing the methods to implement the BST solution. This class contains a main method from which to run the BST solution and is responsible for generating output

Common Classes:

CommonMethods - Static Class containing key generation and key splitting methods to be used by both the LSArrayApp and LSBSTApp classes.

ReadFile – Class To handle file reading in both the LSBSTApp and LSArrayApp classes. Adds data items to the correct data structure based on which class called this class's constructor and allows the LSArrayApp and LSBSTApp classes to populate their data structures.

ScheduleItem - Class to hold the data for a load shedding item, I.e., the key and the areas. Contains methods for accessing data within and instances of this class are placed into the data structures.



Experiment Description

Aim:

The aim of this experiment is to compare the number of operations performed by a BST when compared to an unsorted array by making use of Loadshedding data. The comparison is done by comparing the number of comparisons made when searching through the data structure. As part of the instrumentaion, variables finCount in both the LSArrayApp and LSBSTApp classes are used to count comparisons.Subsets of the data are generated using python scripts and fed into the LSBSTApp and LSArrayApp. Using the subsets, a best, worst and average case is taken from each set and the results are graphed for comparison

Trial Test Values and Outputs(Part 2):	Trial Test Values and Outputs(Part 4):
Input: 1 1 00 Stage: 1, Day: 1, Start Time: 00:00 Areas Affected: 1 Number of operations for find: 1 **Input: 6 29 04** Stage: 6, Day: 29, Start Time: 04:00 Areas Affected: 6, 14, 2, 10, 3, 11 Number of operations for find: 2092 **Input: 8 25 22** Stage: 8, Day: 25, Start Time: 22:00 Areas Affected: 14, 6, 10, 2, 15, 7, 11, 3 Number of operations for find: 2963 **First 10 Lines of PrintAllAreas():** Stage: 1, Day: 1, Start Time: 00:00 Areas Affected: 1 Stage: 1, Day: 17, Start Time: 00:00 Areas Affected: 1 Stage: 1, Day: 2, Start Time: 00:00 Areas Affected: 13 Stage: 1, Day: 18, Start Time: 00:00 Areas Affected: 13 Stage: 1, Day: 3, Start Time: 00:00 Areas Affected: 9 Stage: 1, Day: 19, Start Time: 00:00 Areas Affected: 9 Stage: 1, Day: 4, Start Time: 00:00 Areas Affected: 5 Stage: 1, Day: 20, Start Time: 00:00 Areas Affected: 5 Stage: 1, Day: 5, Start Time: 00:00 Areas Affected: 2 Stage: 1, Day: 21, Start Time: 00:00 Areas Affected: 2 **Last 10 Lines of PrintAllAreas()** Stage: 8, Day: 27, Start Time: 22:00 Areas Affected: 14, 6, 10, 2, 15, 7, 11, 3 Stage: 8, Day: 12, Start Time: 22:00 Areas Affected: 14, 6, 10, 2, 15, 7, 11, 3 Stage: 8, Day: 28, Start Time: 22:00 Areas Affected: 14, 6, 10, 2, 15, 7, 11, 3 Stage: 8, Day: 13, Start Time: 22:00 Areas Affected: 15, 7, 11, 3, 12, 4, 8, 16 Stage: 8, Day: 29, Start Time: 22:00 Areas Affected: 15, 7, 11, 3, 12, 4, 8, 16 Stage: 8, Day: 14, Start Time: 22:00 Areas Affected: 15, 7, 11, 3, 12, 4, 8, 16 Stage: 8, Day: 30, Start Time: 22:00 Areas Affected: 15, 7, 11, 3, 12, 4, 8, 16 Stage: 8, Day: 15, Start Time: 22:00 Areas Affected: 15, 7, 11, 3, 12, 4, 8, 16 Stage: 8, Day: 31, Start Time: 22:00 Areas Affected: 15, 7, 11, 3, 12, 4, 8, 16 Stage: 8, Day: 16, Start Time: 22:00 Areas Affected: 15, 7, 11, 3, 12, 4, 8, 16	**Input: 1 10 00** Stage: 1, Day: 10, Start Time: 00:00 Areas Affected: 15 Number of insert operations: 278192 Number of find operations: 5 **Input: 6 25 18** Stage: 6, Day: 25, Start Time: 18:00 Areas Affected: 12, 4, 8, 16, 13, 5 Number of insert operations: 278192 Number of find operations: 279 **Input: 8 9 22** Stage: 8, Day: 9, Start Time: 22:00 Areas Affected: 14, 6, 10, 2, 15, 7, 11, 3 Number of insert operations: 278192 Number of find operations: 407 **First 10 Lines of printAllAreas():** Stage: 1, Day: 10, Start Time: 00:00 Areas Affected: 15 Stage: 1, Day: 10, Start Time: 02:00 Areas Affected: 16 Stage: 1, Day: 10, Start Time: 04:00 Areas Affected: 1 Stage: 1, Day: 10, Start Time: 06:00 Areas Affected: 2 Stage: 1, Day: 10, Start Time: 08:00 Areas Affected: 3 Stage: 1, Day: 10, Start Time: 10:00 Areas Affected: 4 Stage: 1, Day: 10, Start Time: 12:00 Areas Affected: 5 Stage: 1, Day: 10, Start Time: 14:00 Areas Affected: 6 Stage: 1, Day: 10, Start Time: 16:00 Areas Affected: 7 Stage: 1, Day: 10, Start Time: 18:00 Areas Affected: 8 **Last 10 lines of printAllAreas():** Stage: 8, Day: 9, Start Time: 04:00 Areas Affected: 5, 13, 1, 9, 6, 14, 2, 10 Stage: 8, Day: 9, Start Time: 06:00 Areas Affected: 6, 14, 2, 10, 7, 15, 3, 11 Stage: 8, Day: 9, Start Time: 08:00 Areas Affected: 7, 15, 3, 11, 8, 16, 4, 12 Stage: 8, Day: 9, Start Time: 10:00 Areas Affected: 8, 16, 4, 12, 9, 1, 5, 13 Stage: 8, Day: 9, Start Time: 12:00 Areas Affected: 9, 1, 5, 13, 10, 2, 6, 14 Stage: 8, Day: 9, Start Time: 14:00 Areas Affected: 10, 2, 6, 14, 11, 3, 7, 15 Stage: 8, Day: 9, Start Time: 16:00 Areas Affected: 11, 3, 7, 15, 12, 4, 8, 16 Stage: 8, Day: 9, Start Time: 18:00 Areas Affected: 12, 4, 8, 16, 13, 5, 9, 1 Stage: 8, Day: 9, Start Time: 20:00 Areas Affected: 13, 5, 9, 1, 14, 6, 10, 2 Stage: 8, Day: 9, Start Time: 22:00 Areas Affected: 14, 6, 10, 2, 15, 7, 11, 3

Table 1: Successful Trials

Invalid Array Inputs	**Invalid BST inputs**
input: 12 4 24 Stage: 12, Day: 4, Start Time: 24:00 Areas Affected: Areas not Found Number of operations for find: 2976 **Input: 4 24** Your input should be of the format xx yy zz **Input: 1 15 28** Stage: 1, Day: 15, Start Time: 28:00 Areas Affected: Areas not Found Number of operations for find: 2976	**Input: 2 56 4** Areas not Found Number of insert operations: 278192 Number of find operations: 72 **Input: 2 4** Your input should be of the format xx yy zz **Input: 2 4 91** Areas not Found Number of insert operations: 278192 Number of find operations: 72

Table 2: Unsuccessful trials

Creativity In Application

Creativity comes in with the ReadFile class. Instead of including file reading functionality into the LSBSTApp and LSArrayApp classes, ReadFile checks the stack to determine which class called it, trigerring a flag that signals which data structure the data should be added to. Since both the LSArray and LSBST require the making an breaking of keys from the args passed into the main methods, a helper class (CommonMethods) was created to avoid rewriting code. Python scripts were used to create subsets of data from the load shedding schedule as well as to automate their run. Furthermore UML class diagrams were used to show relationships between classes.

Summary of git usage(empty log lines omitted from first and last 10)

From Terminal:

0: commit 451d96f0d4c025c06435cb6cfa3c27334eec3781

1: Author: VoidMaster23 <edsonshivuri@gmail.com>

2: Date: Thu Mar 5 05:40:50 2020 +0200

4: Tests Done

6: commit 0ea9a42742fd1f8a2cc665ff643a5f4add014cbd

7: Author: VoidMaster23 <edsonshivuri@gmail.com>

8: Date: Thu Mar 5 01:36:20 2020 +0200

10: code done, report left

…

84: commit b2e4e0dedc5f0f765e60ea6529698d2cc1abb2ee

85: Author: VoidMaster23 <edsonshivuri@gmail.com>

86: Date: Mon Mar 2 00:59:15 2020 +0200

88: LSArrayApp file reading done, must finish and do LSBSTApp

90: commit 8deb14ba2373f264b01d52b4d0272909c2beb452

91: Author: VoidMaster23 <edsonshivuri@gmail.com>

92: Date: Sat Feb 29 20:37:42 2020 +0200

94: First commit, ScheduleItem class done (might need tweaking)

Graphs and Discussion

Note: for this experimen, only find operations were taken to account for each data set, however it is important to remember that for arrays , as long as you have the index, insertion is constant time. Meanwhile, to insert into a BST, a comparison is made for each insertion to put the item in the correct place, hence large count values as seen below:

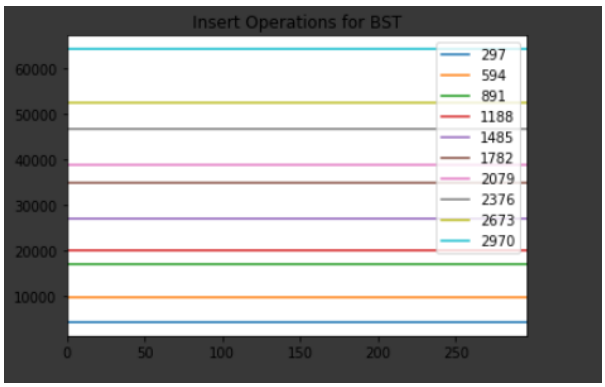


Figure 1: BST Insert Operations

Best Case Graph

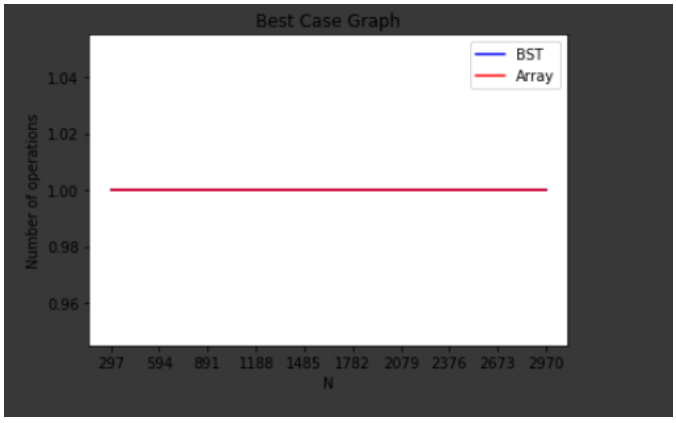


FIgure 2: Best Case Graph

or both data structures, looking for the first item will always result in in an operation count of one as the only comparison needed is to check if the first array element/BTNode is the one we are looking for. Since the data is unsorted, the first element would be our best case. Hence for both structures , the best case is O(1).

Worst Case Graph

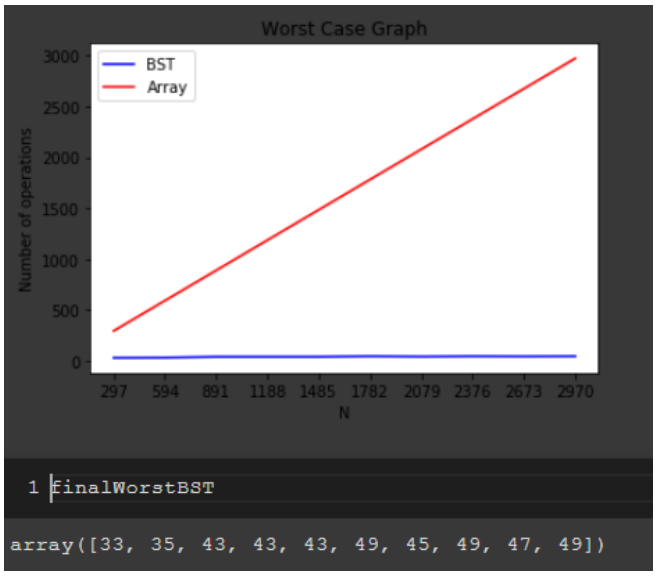


FIgure 3: Worst Case Graph

The worst case graphs show that for the array, the worst case would be searching for an element at the very end, leading to an O(n) search time. This is seen by the linear relationship between N and the number of operations. Looking at the BST, however we see the worst case values remaining relatively constant in comparison. In the worst case, A BST will be O(log n)

Average Case Graph

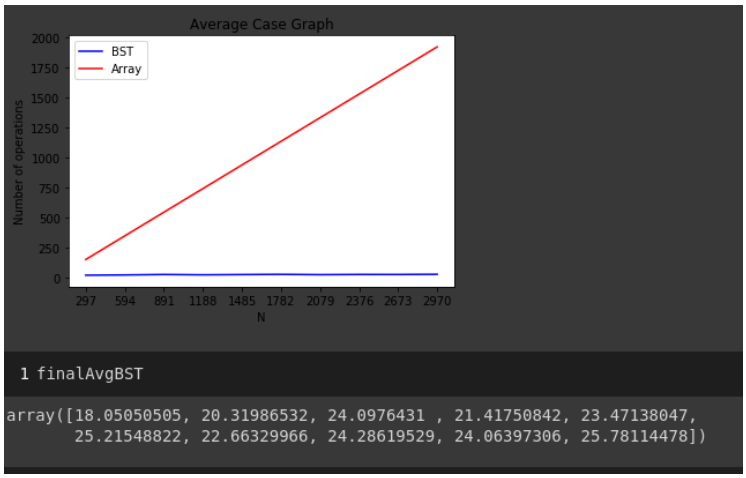


FIgure 4: Average Case Graph

The average case was obtained by determining the average of all operation counts across the different data sets. From the above, we can see that the average case plots have the same relationship ad the worst case. On average, the array is O(n) while the BST is O(log n)

Conclusions

From the above discussion, it is to be concluded that for large data sets, the BST is more efficient for searching, however a high insertion count makes it more practical for when searching is the main goal. For smaller data sets (<=100 entries), the data structures behave similarly and thus an array can efficiently be used with them.