STHZAK001

Zakhele Sithole

CSC2001F: Assignment 2 Report

Part 1 and Part 3

Starting with Part 1, 2, 3 and 4 we are going to be using two types of trees: AVL Tree and Binary Search Tree.

AVL Tree:

With AVL trees there are two rotations involved in the algorithm and then you also get the double rotations, for the tree to be unbalanced, it must at-least have a height of 2. This is a Java Program to implement AVL Tree. An AVL tree is a self-balancing binary search tree, and it was the first such data structure to be invented. In an AVL tree, the heights of the two child subtrees of any node differ by at most one; if at any time they differ by more than one, rebalancing is done to restore this property. Lookup, insertion, and deletion all take O (log n) time in both the average and worst cases, where n is the number of nodes in the tree prior to the operation. Insertions and deletions may require the tree to be rebalanced by one or more tree rotations namely: left rotation, right rotation and left and right rotation.

Binary Search Tree

It consists of the following, each node has a key, the key in the left subtree are less than the key in its parent node, the key in the right are greater than the key in its parent node and it cannot generate duplicated nodes. There are several operations on BST's that are important to understand. We will discuss some of the basic operations such as how to insert a node into a BST, how to delete a node from a BST and how to search for a node in a BST.

Both These Trees Consist of The Following Functions:

Insert Function: which starts from the root node, if the node to insert is less than the root, we go to left child, and otherwise we go to the right child of the root.

Find Function: which starts from root node, and then go to the left subtree or right subtree depending on the conditions I have and if the search data meets hem or not until we find (or not find the node). If the node is equal to root, then we return true. If the root is null, then we return false. If the root is null.

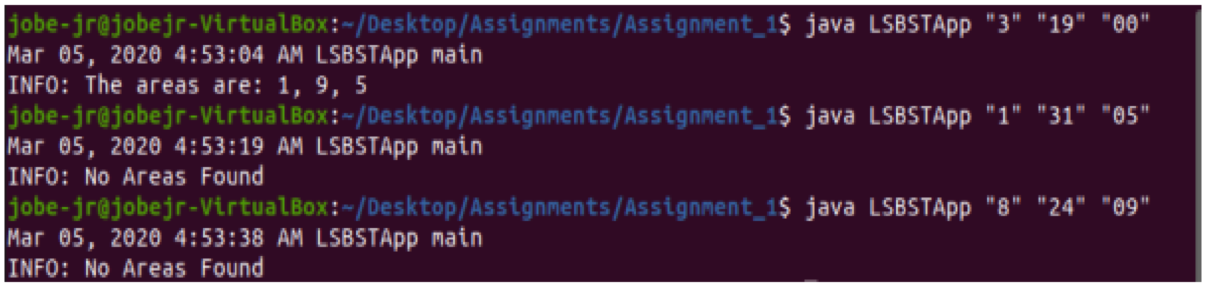
Delete function: This is a function that uses the find method as in need to find the specific node or child that needs to be deleted form the tree in order for that to happen, we need to find out first if that node or child exist then do the deletion.

This classes also included methods like printAreas () used to print areas of a specific slot which consist of two arguments start Time which is a string of the time at which load shedding starts, day at which the load shedding is planned to occur and lastly stage which is the level at which the load shedding will be set to be on. which uses a string as an argument to be compared with the data, that incrementing the number of counts it took to find it in the list, we are having from the txt file and then printing out the corresponding Stage, day and startTime.

printAllAreas() a method used to print all arear within the txt file.

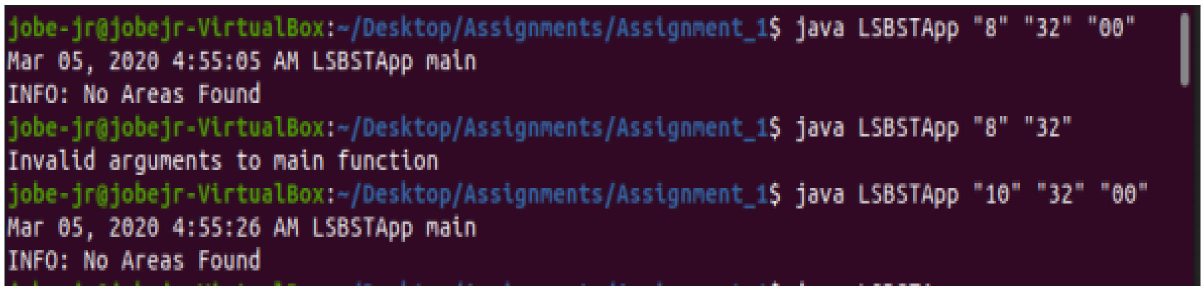
Part 2 and Part 4

Results Form Binary Search Tree



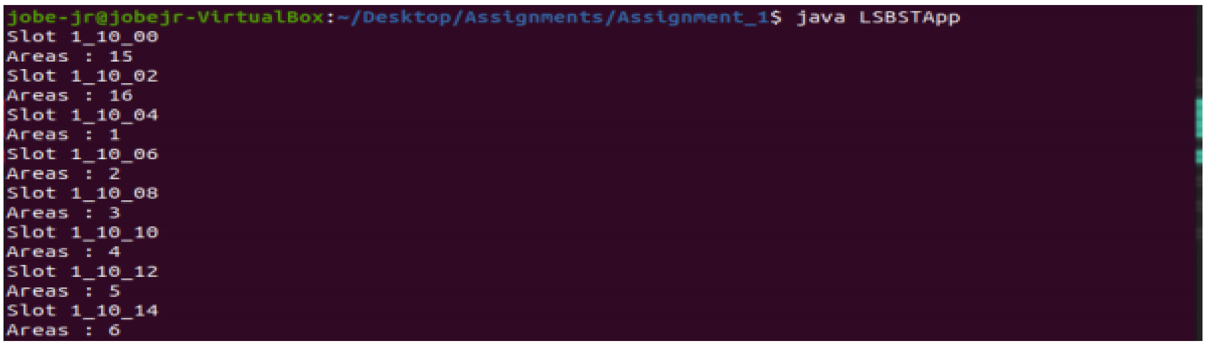
```
jobe-jr@jobejr-VirtualBox:~/Desktop/Assignments/Assignment_1$ java LSBSTApp "3" "19" "00"
Mar 05, 2020 4:53:04 AM LSBSTApp main
INFO: The areas are: 1, 9, 5
jobe-jr@jobejr-VirtualBox:~/Desktop/Assignments/Assignment_1$ java LSBSTApp "1" "31" "05"
Mar 05, 2020 4:53:19 AM LSBSTApp main
INFO: No Areas Found
jobe-jr@jobejr-VirtualBox:~/Desktop/Assignments/Assignment_1$ java LSBSTApp "8" "24" "09"
Mar 05, 2020 4:53:38 AM LSBSTApp main
INFO: No Areas Found
```

3 known parameter combination that works



```
jobe-jr@jobejr-VirtualBox:~/Desktop/Assignments/Assignment_1$ java LSBSTApp "8" "32" "00"
Mar 05, 2020 4:55:05 AM LSBSTApp main
INFO: No Areas Found
jobe-jr@jobejr-VirtualBox:~/Desktop/Assignments/Assignment_1$ java LSBSTApp "8" "32"
Invalid arguments to main function
jobe-jr@jobejr-VirtualBox:~/Desktop/Assignments/Assignment_1$ java LSBSTApp "10" "32" "00"
Mar 05, 2020 4:55:26 AM LSBSTApp main
INFO: No Areas Found
```

3 known parameter combination that are invalid



```
jobe-jr@jobejr-VirtualBox:~/Desktop/Assignments/Assignment_1$ java LSBSTApp
Slot 1_10_00
Areas : 15
Slot 1_10_02
Areas : 16
Slot 1_10_04
Areas : 1
Slot 1_10_06
Areas : 2
Slot 1_10_08
Areas : 3
Slot 1_10_10
Areas : 4
Slot 1_10_12
Areas : 5
Slot 1_10_14
Areas : 6
```

No Parameters Passed In/No Entries

Results from AVL Tree

Conclusion:

The find operation:

The AVL Tree is much faster than BST because AVL Trees are self-balancing trees and so the height is as small as possible. So, the element is found immediately

The Insertion Operation:

The insertion of elements in sorted order in AVL Trees is faster than on BST. This is dependent upon the order that the elements are added in whether its ascending or descending order, the elements are added either on left or on the right side. When more than two elements are added, rebalancing is done which results in the decrease on the height of the tree.

The insertion of elements in BST are added from one side, which leads to the tree being heavy on one side, whenever a new element is added into the tree it must go further down to reach the end of the tree

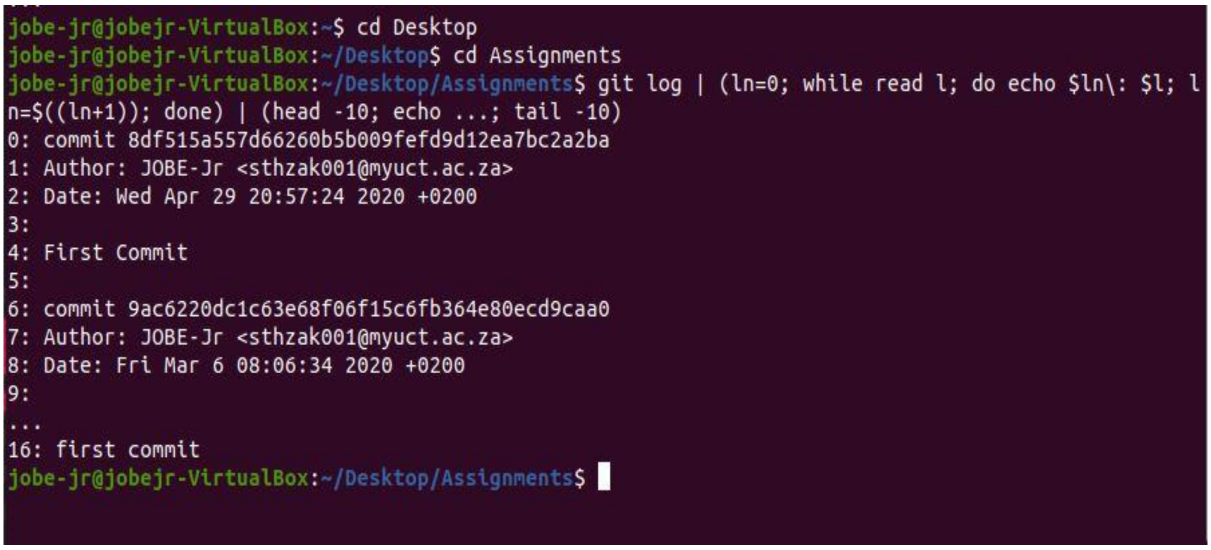
The AVL Tree is faster than the BST. When we know that we must do more searching than insertions, we should use AVL Trees to accomplish the Task. The Insertions into AVL trees may take more time than in the Binary Search Tree when random elements are inserted, but the difference in the time taken is not that quite significant as it is about 60 Seconds rough estimate. Into the real world the AVL Trees has more practical applications than the BST because of its balancing property.

How to run:

To run the code or programs you need to go to the src folder on the terminal then type in java “name-of-program” “input” press enter to run the program o then this will return corresponding areas for that requested stage, date and startTime.

If there’s no input, it will print out all the areas.

Git Log



```
...
jobe-jr@jobejr-VirtualBox:~$ cd Desktop
jobe-jr@jobejr-VirtualBox:~/Desktop$ cd Assignments
jobe-jr@jobejr-VirtualBox:~/Desktop/Assignments$ git log | (ln=0; while read l; do echo $ln\: $l; l
n=$((ln+1)); done) | (head -10; echo ...; tail -10)
0: commit 8df515a557d66260b5b009fefd9d12ea7bc2a2ba
1: Author: JOBE-Jr <sthzak001@myuct.ac.za>
2: Date: Wed Apr 29 20:57:24 2020 +0200
3:
4: First Commit
5:
6: commit 9ac6220dc1c63e68f06f15c6fb364e80ecd9caa0
7: Author: JOBE-Jr <sthzak001@myuct.ac.za>
8: Date: Fri Mar 6 08:06:34 2020 +0200
9:
...
16: first commit
jobe-jr@jobejr-VirtualBox:~/Desktop/Assignments$ 
```