CSC2001F 2020 Assignment 1

Introduction

We have been tasked with the challenge of implementing 2 different data structures onto a load shedding text file in order to test which of the 2 data structures is faster for searching through the text file and also to observe which of 2 is better for insertion. The 2 data structures which we shall be comparing are an array of objects and the other is a Binary search tree.

OO design and Classes

My OO design for the assignment consists of 9 classes in total. I have created 2 classes for the LSArrayApp program which are LSArrayApp and LSA (which is short for Loadshedding array). I created 2 classes for the LSBSTApp which are: LSBSTApp and TextF (which is almost the same class as LSA but slightly different) on top of implementing the BinarySearch tree classes which were provided to us on vula so in total for LSBSTapp there are 7 classes for LSBSTApp.

LSA

The purpose of this class is to create an object for the array of objects class. It stores all your data that has been extracted from your text file. The information which is extracted from the text file in each line is the key which consists of the stage, day and time and the areas affected. So like any other object class you will have functions and methods which return the information you need to your main class. These methods are:

- The constructor: which instantiates your variables for the key and the area and define the parameters.
- Getstage(), getTime(), getDay() and getAreas() which allows the main class to access the data stored in the object correctly
- toString(): just converts all your data to string format

this object is then used in the main class to create the object of arrays

LSArrayApp

This is the main class for the LSArrayApp program. The attributes which are instantiated at the beginning of the class are:

- the Arrayobj variable which creates the array of objects of size 2976
- the opCount variable to count the number of comparisons in the program

The programs consist of 5 methods which are:

- The main method which is entry point of your program. In this main method we call the methods printallareas or printareas depending on how many arguments were inputted
- The textfile method which extracts the textfile and separates each line into its key and areas affected. To prevent an IO Exception, the entire process is placed within a try-catch statement. Once the key and areas are found it is then placed in the array of objects to be stored
- printAreas which prints all the data stored in your array
- printAllAreas which prints the specific areas when the user enters the key and prints the number of comparison which done searching for that key and area

TextF

The TextF class is almost identical to the LSA app except there are a few methods which I have changed and added to make it more suitable for the Binary search data structure. I modified the constructor so now it does the string manipulation for the textfile in this the object and also implements a compareTo method which is later used in your binary search tree class to help with your find method. Those are the only differences between TextF and LSA

LSBSTApp

The same methods and attributes are used as in the LSArrayApp but with minor changes to the program to account for the binary search tree data structure. So now the Textfile method has been shortened as the string manipulation has taken place in the TextF class. Now instead of inserting into the array of objects we are inserting into the binary search tree using the .insert method. For the printAreas method instead of incrementing the counter inside that method we instead increment the counter in the binary search tree

Binary search tree classes

The binary search tree classes were all of generic data type and were provided to us on vula. I slightly modified the Binary search tree class and the binary tree node class. In the binary search class I added in a counter in insert and find classes for my comparison counter that is need for the LSBSTApp and in the binary tree node class I added in a getData class to return data of one specific node to my LSBSTApp.

Experiment description

This experiment is aimed at comparing the implementation of the Binary Search Tree with that of the traditional array object data structure. This will be practically measured through the comparison operation counts made when each data structure is implemented. The comparison counts are incremented each time the user attempts to enter the key value to find a specific value.

In order to solve this problem I have made .java files that extract the contents from the textfile and store it in it's respective data structure. The next step was to setup the comparison counter. For both data structures the counter is implemented when searching through the data structure for the data item.

Once the searching is complete the number of comparisons is recorded. If the search was successful the data item alongside the comparions will be displayed if the search was unsuccessful then a message displaying that that search for your data was not found and the number of comparisons made will be displayed.

The next step I took was to find the best/worst/average cases for the comparison. So I create a python program which creates 10 texts files of varying size which are then put through the LSBSTApp and LSArrayApp. So my output from those programs will be then sent to another textfile to record the cases for best/average/worst case. This data is then translated to graph form for further interpretation. Before doing any of this we already know the binary search tree is superior so all we're doing is proving this fact.

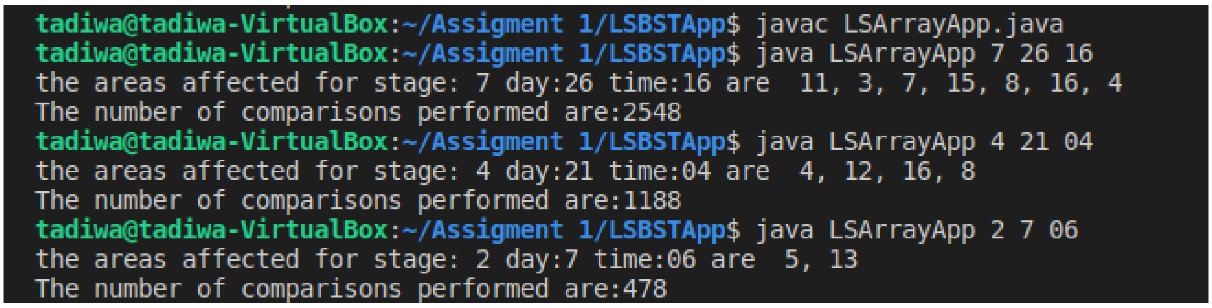
Trial test value and outputs

Part 2

The three correct parameters I inputted for part 2 were:

- 7 26 16
- 4 21 04
- 2 7 06

The output was as follows:

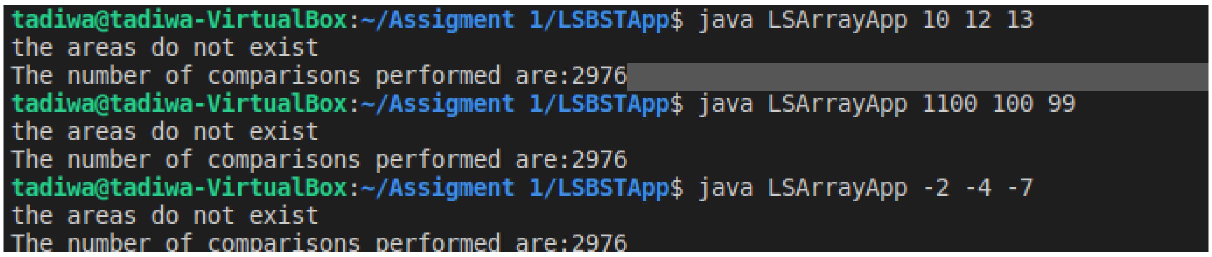


```
tadiwa@tadiwa-VirtualBox:~/Assigment 1/LSBSTApp$ javac LSArrayApp.java
tadiwa@tadiwa-VirtualBox:~/Assigment 1/LSBSTApp$ java LSArrayApp 7 26 16
the areas affected for stage: 7 day:26 time:16 are  11, 3, 7, 15, 8, 16, 4
The number of comparisons performed are:2548
tadiwa@tadiwa-VirtualBox:~/Assigment 1/LSBSTApp$ java LSArrayApp 4 21 04
the areas affected for stage: 4 day:21 time:04 are  4, 12, 16, 8
The number of comparisons performed are:1188
tadiwa@tadiwa-VirtualBox:~/Assigment 1/LSBSTApp$ java LSArrayApp 2 7 06
the areas affected for stage: 2 day:7 time:06 are  5, 13
The number of comparisons performed are:478
```

The three incorrect values were :

- 10 12 13
- 1100 100 99
- -2 -4 -7

The output was as follows:



```
tadiwa@tadiwa-VirtualBox:~/Assigment 1/LSBSTApp$ java LSArrayApp 10 12 13
the areas do not exist
The number of comparisons performed are:2976
tadiwa@tadiwa-VirtualBox:~/Assigment 1/LSBSTApp$ java LSArrayApp 1100 100 99
the areas do not exist
The number of comparisons performed are:2976
tadiwa@tadiwa-VirtualBox:~/Assigment 1/LSBSTApp$ java LSArrayApp -2 -4 -7
the areas do not exist
The number of comparisons performed are:2976
```

<u>**Part 4**</u>

The same tests with the same values were conducted for part 4 using the binary search and the following results were found:

- Areas: [11, 3, 7, 15, 8, 16, 4]
- Number of comparisons: 41
- Areas: [4, 12, 16, 8]
- Number of comparisons: 23
- Areas: [5, 13]
- Number of comparisons: 11

For the incorrect parameters the following was found:

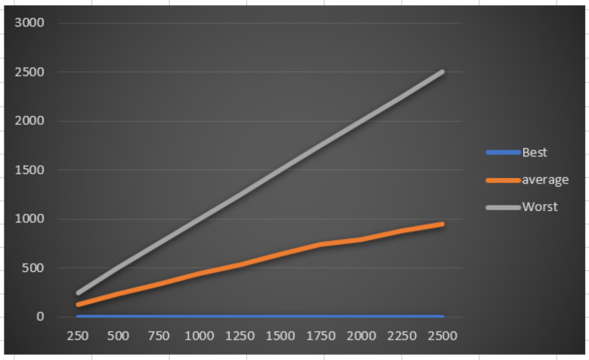
- This area does not exist
- Number of comparisons: 51
- This area does not exist

- Number of comparisons: 51
- This area does not exist
- Number of comparisons: 51

Results part 5

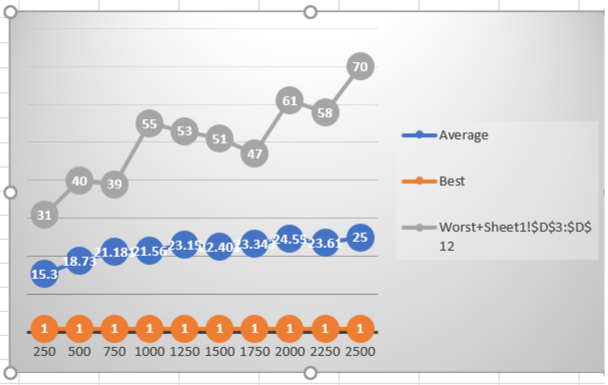
The results in graph and table form of the array data structure is as follows:

Sample(n	best	Average	worst
250	1	123.396	250
500	1	238.5	500
750	1	339.1466	750
1000	1	445.548	1000
1250	1	530.3436	1250
1500	1	643.122	1500
1750	1	740.444	1750
2000	1	789.193	2000
2250	1	880.908	2250
2500	1	945.44	2500



The result in graph and table form of the Binary search tree is as follows:

mple(n)	best	average	worst
250	1	15.3	31
500	1	18.73	40
750	1	21.181	39
1000	1	21.56	55
1250	1	23.15	53
1500	1	22.401	51
1750	1	23.343	47
2000	1	24.55	61
2250	1	23.61	58
2500	1	25	70



Discussion and results

From the graph of the array objects it shows number of comparisons for a data item or for the size of data(N). The graph is a linear one, showing that the time complexity for an array is O(N). This would reveal that as the data set increases, the corresponding number of comparisons increase. That is to say, if one is searching for the 2490th item, the array would traverse through the first 2489 items until it reached 2490. Which was expected for the array

The graph for the BST was not expected and it shows that the relationship is not easily pronounced and may require a larger dataset to truly show the logarithmic pattern. Furthermore, this may mean that the BST class may not be accurate enough to truly reveal the binary tree properties.

GIT USAGE LOG

1. `commit da90d11065d8fd79c0bb26a8532273b926b3c518 (HEAD -> master, origin/master)`
2. `Author Tadiwanashe Muzondo <tmuzondo4@gmail.com>`
3. Date: Tuesday Mar 3 18:06:28 2020 +0200