CSC2001F

Assignment 2

Experiment Description

This assignment is about testing the efficiency of AVL Trees, A CVS file with a list of 9919 entries of country names, dates and vaccination numbers. The general idea was to check if a country has had vaccines done on a particular date and how many vaccinations were completed on that date. The data is stored in an array first, randomized and then store in an AVL Tree after that. The data that is stored in these data structures is read from the provided CSV file and the file is broken down into 3 separate arrays of the country name, the date and the number of vaccinations. Various functions and methods are used to make use of the data provided. A combination of the country and the date is used to find vaccinations numbers of a particular day, that is however not necessarily the focus of this assignment, however this combination will still be utilized to control the data more efficiently.

The aim of this assignment is to check if AVL Trees really do balance the nodes and provided good results regardless of the order of the data. The best way to do this is to randomize the data and test the AVL Tree across different variations of the randomized data. This serves that purpose by randomizing the data using the Random class from Java as a sure way to make the data randomized, not only that but the randomization will be repeated x times to make sure it is consistent across all the x times.

Randomization Approach & Application Design:

The randomization approach of this assignment revolves around using the Random class to randomize the values in the AVL Tree, across all the maximum of 20 different tests or permutations of the AVL Tree. Basically the AVL Tree can be randomized x times, each time the output is displayed and written to a File called RandomizedFile, the RandomizedFile will contain x amount of files, each containing the output for that specific randomization. So all the x files have different data that has been randomized each time and then sent to these files. In each of these files, below is the minimum, average and maximum number of values for that specific permutation of the data.

VaccineArray The class VaccineArray.java is a very short class used by VaccineArrayApp.java. VaccineArray stores the accessor methods to retrieve information about the entries in the csv file. It has methods like getCountry() which returns the country name. It also has a printAll() method which returns the country name along with its vaccination numbers for a specified date. The constructor will be used in the VaccineArrayApp class and the BSTArrayApp as well to create an object or instance of the class VaccineArray. This is to be able to access each country’s name and vaccination numbers. This is going to be utilized by the printResults() and printComparisons(). Class: VaccineArrayApp The class VaccineArrayApp contains the main class and most of the methods for part 1. This class creates an object of VaccineArray to make use of the methods in the VaccineArray class. Methods in VaccineArrayApp: 1. printAllresults(String place, String date)– This method is used to search for all the ‘country + date’ combinations that match with the supplied data (arguments) in the traditional array that they are stored in. The data received is then captured in a text file. The textfile format is the country name and the vaccination numbers. 2. printComparisons(String place, String date) - This method is used to determine the number of iterations that are needed to find a specified combination of country + date. This is a way to determine to determine the efficiency of the traditional array, by determining how many iterations are required for a data entry, the larger the overall iterations are, they less efficient the data structure being used is. 3. calculatesubsets(int subset, String sDate, PrintWriter sampledata) – This method is used to test the array using 10 different subsets of the original sample size that are spaced equally apart. For each of the 10 subsets, the number of iterations that are needed to find the county + date combination is determined. Using these iterations, the best case, worst case and the average case is determined for each of the 10 subsets of values and it is written to a textfile each time. The method takes in the subset, the date, and the file name to be written to as arguments. 4. createFiles(String date) – This method is used to create 10 different text files named sample1.txt to sample10.txt to store the results for the 10 different subsets. The method then calls the calculatesubsets method to write the results to the 10 different text files for the 10 different subsets There is only one argument, it is the date. Class: VaccineBSTApp The AccessBSTApp class is used for part three, four and five of the assignment. Information about students is now stored in a different data structure which is a BST (Binary Search Tree). This class also makes use of multiple other classes and creates objects of those classes, which are The BinarySearchTree class, BinaryTree class and BinaryTreeNode class. Methods in class VaccineBSTApp: 1. printAllresults(String place, String date)– This method is the same as the one used in VaccineArrayApp except for the fact that it is used to accommodate the Binary Search Tree instead of the Array. 2. printComparisons() - 3. calculatesubsets() - 4. createFiles() –

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ Array test values and Outputs: \*This is a sample from the textfile sample6.txt from Arraytextfile folder\* There are 9342 iterations for Norway There are 195 iterations for Mongolia There are 899 iterations for Senegal There are 4453 iterations for Madagascar There are 2286 iterations for Trinidad and Tobago There are 8972 iterations for Hungary There are 1463 iterations for Curacao There are 7444 iterations for Guatemala There are 987 iterations for Syria There are 9904 iterations for Cyprus There are 5356 iterations for Chile There are 5774 iterations for Tunisia There are 2286 iterations for Trinidad and Tobago There are 8818 iterations for Turks and Caicos Islands There are 1063 iterations for Netherlands The average case is 5071 |The worst case is 9918 |And the best case is 3 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ BST test values and Outputs: \*This is a sample from textfile sample9.txt\* There are 7 iterations for Croatia There are 10 iterations for Cyprus There are 16 iterations for Israel There are 17 iterations for Kuwait There are 10 iterations for Barbados There are 6 iterations for Fiji There are 13 iterations for Moldova There are 13 iterations for Asia There are 4 iterations for Peru There are 7 iterations for Yemen There are 7 iterations for Egypt There are 8 iterations for Ukraine There are 9 iterations for Zimbabwe There are 10 iterations for France There are 18 iterations for Kazakhstan There are 14 iterations for Maldives The average case is 6 |The worst case is 55 |And the best case is 1 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ Results: \*This is data that is extracted from different sample files but showcasing the same countries\* Country Array iterations BST iterations Cyprus 9904 19 France 7568 10 England 4497 5 Zimbabwe 6572 9 Netherlands 1063 18 Senegal 899 6 Chile 5356 3 Maldives 9918 14 Croatia 5378 7 Canada 126 5 United States 400 21 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_ Discussion of results: As we can see from the above results of iterations, the BST iterations are much lower in all the iterations chosen and they seem to take a logarithmic shape. The array has to loop through every item every time it is looking for a value in the dataset, this makes the iterations for each data set to be very large and the array is linear.The formula to define the graph to draw the iterations of the Array comparisons would be f(x)=mx +c and that of the BST would be f(x)=log(x). The above results also tell us that it takes less time to search through the BST data structure than an array data structure. For the first values of x when we plot them in our formula. We would get approximately the same values. But when the x value increases the linear function grows a lot compared to the log function. Hence the time it will take to do comparisons in the Array will be bigger compared to time it will take when using a binary search system. Examples of how the graphs would look like: Git Usage: Conclusion: The conclusion is therefore that a BST is way more efficient to use when you have a lot of data compared to an Array. It takes less time to use a BST data Structure because it is O(log n) and linear is O(n). And as the dataset size increases, so does the inefficiency of the Array data structur