CSC2001F Assignment 1 Report

Done by Taahir Suleman – SLMTAA007

The Assignment and programs

I created programs to receive input from a vaccinations.csv file containing multiple instances of a country's vaccinations on a particular date, in the order country, date, vaccinations. I did this in two ways: using an array of objects and a Binary Search Tree (BST), with each case containing Vaccine objects with the country, date and vaccinations specified above.

My Object Oriented Design

Firstly, I created a program shared by both the array of objects and BST methods called Vaccine.java that contains instance variables for the country, date and vaccinations, storing them all as strings. Thus, in both cases, this class is used to create Vaccine objects containing the 3 properties from each line of vaccinations.csv. This program had two constructors: One that takes in a String parameter containing a line from the file, which then splits up this line into the 3 String properties, and this constructor is used to create Vaccine objects when reading from the file. The second constructor takes in String parameters for country and date, which is used to create temporary Vaccine objects using a date and country combination as entered by the user when the program is run, which is then used by the relevant find methods in each solution case. Lastly, this program contains a compareTo method which compares two keys of Vaccine objects, with a key being a concatenated String of the country and date of each Vaccine, and returns an integer which returns 0 if the keys match, and this method is used when finding Vaccine objects that match the details entered by the user when the program is run

Array of objects solution:

I created a class called VaccineArray.java that creates a Vaccine array of objects of size 10 000 and continuously appends Vaccine objects to this array every time the addVax() method is called. This method is used by the other class for this solution method which is called VaccineArrayApp.java. The readData() VaccineArrayApp uses this addVax() method to add a Vaccine object to the array in VaccineArray as it reads each line of the vaccinations.csv file, using the single parameter Vaccine constructor mentioned above to create these objects. Secondly, the findVax() method in VaccineArray is used by the userInteraction()

method in VaccineArrayApp to search through the Vaccine array of objects and compare each object's key (as specified above) with that inputted by the user until either a Vaccine object is found that matches this key or the end of the array has been reached. Lastly, the VaccineArray class has an outputVax() method which returns a string array containing elements that either contain the vaccination quantity or "<Not found>" for each country: this is then used by the userInteraction() method in the VaccineArrayApp to output each country's (that were inputted by the user) vaccination numbers on the given date. VaccineArrayApp contains the main method which calls the readData() and userInteraction() methods from a VaccineArrayApp calling object created in the main method: Thus, the main method allows for the vaccinations.csv to be read and for the respective Vaccine objects to be created; facilitate user input; use this user input to find the Vaccine object that matches the key inputted with a method in VaccineArray that uses the compareTo() method in Vaccine; and lastly output the country and vaccination number for this Vaccination object. This is how the Vaccine, VaccineArray and VaccineArrayApp interact for this solution.

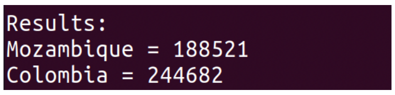
BST solution:

Firstly, I used the BinarySearchTree and its associated classes as provided by prof. Hussein Suleman. This class can be used to create a BST and use the necessary operations to manipulate it. From this class the primary operations I utilized were the insert and find operations with their respective methods. I created a class named VaccineBST that first creates a BST object for the Vaccine class (since the BST class uses a generic data type placeholder). In this VaccineBST class the extractData() method is used to read from vaccinations.csv, creating a Vaccine object for each line (using the single parameter constructor) and adding each of these objects to the BST using its insert() method. Secondly, the findVax() method uses the BST's find() method to search through the BST for a key inputted by the user and return either null or the Vaccine object depending on whether a match is found in the BST. Secondly, I created a class named VaccineBSTApp that uses the extractData() method mentioned above to read from the vaccinations.csv and populate the BST accordingly. Furthermore, there is an output() method in VaccineBSTApp that facilitates input; uses the aforementioned findVax() method to search for user-inputted keys and returns either a found Vaccine object or null and then lastly outputs the vaccination numbers accordingly. VaccineBSTApp has a main method that calls output() and extractData() to perform this solution accordingly. Thus, the Vaccine, BinarySearchTree, VaccineBST and VaccineBSTApp classes interact to make this solution effective.

Experimentation

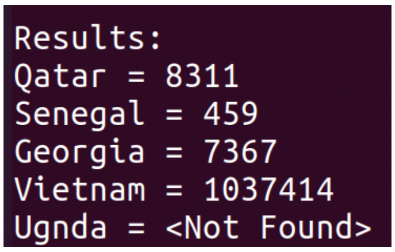
I performed two parts of experimentation on each of these solutions. Part 1 entailed testing these two applications with differing data sets of differing sizes to ensure that both are working as expected. The goal of this experiment was to determine the efficacy of VaccineArrayApp and VaccineBSTApp when provided with differing data sets that cover varying cases of input of different sizes. I performed this experiment by using input redirection to a file containing these test input data sets. Furthermore, I used output redirection to append the results of each trial to a text file for later inspection. In each case, VaccineArrayApp and VaccineBSTApp behaved in the same manner and produced the same output.

I used the same 3 lists of data to test each program and they can be summarized as follows:
The first list of input was: 2022-01-08 with the countries Mozambique and Colombia this produced the expected output as follows:



```
Results:
Mozambique = 188521
Colombia = 244682
```

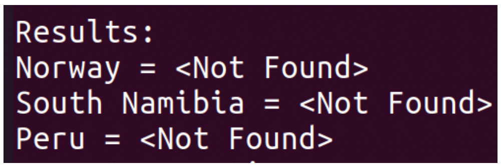
This input data contained a date and two countries which corresponded to data items in the vaccinations.csv file and produced output as expected, showing that the VaccineArrayApp and VaccineBSTApp work as expected with only input that exists in the file. The second list was: 2022-01-02 with the countries: Qatar; Senegal; Georgia; Vietnam and Ugnda. This produced the following output:



```
Results:
Qatar = 8311
Senegal = 459
Georgia = 7367
Vietnam = 1037414
Ugnda = <Not Found>
```

This set of input contained a valid date and 4 valid countries with one country name (Ugnda) misspelt and thus invalid (doesn’t exist in vaccinations.csv). This accounts for the case of their being mixed input with valid and invalid entries and it produced output correctly and as expected, proving that VaccineArray and VaccineBST work with this type of input.

The third and last set of input was: 2022-05-06 with the countries Norway, South Namibia and Peru and produced the following output:



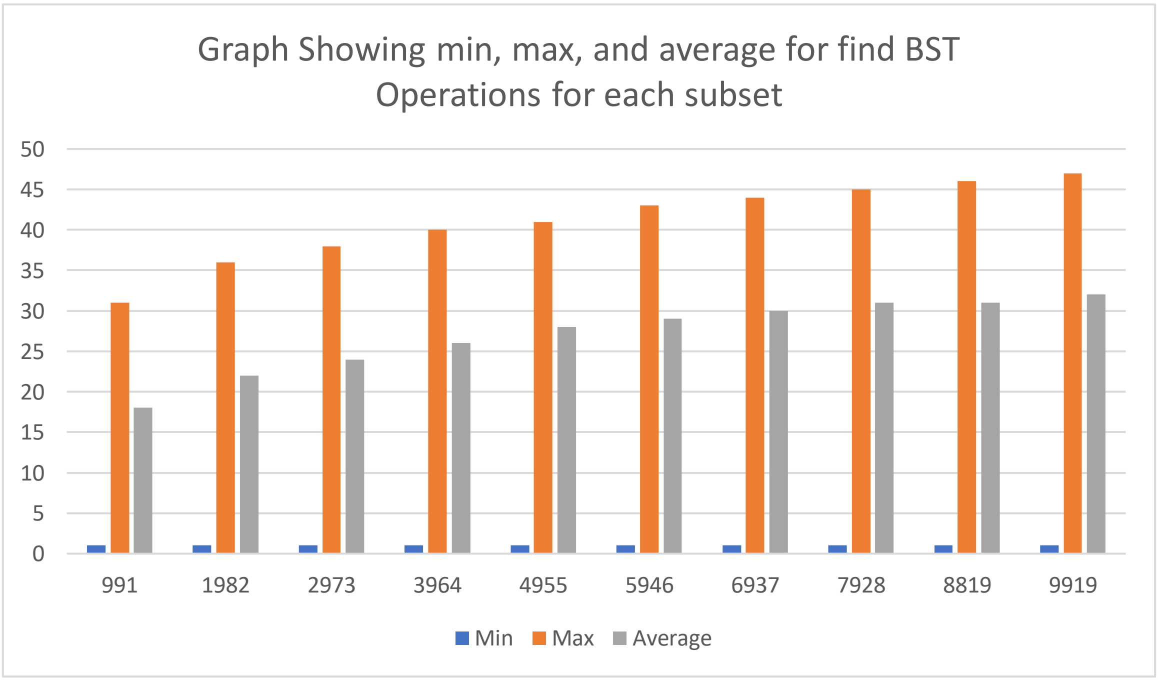
```
Results:
Norway = <Not Found>
South Namibia = <Not Found>
Peru = <Not Found>
```

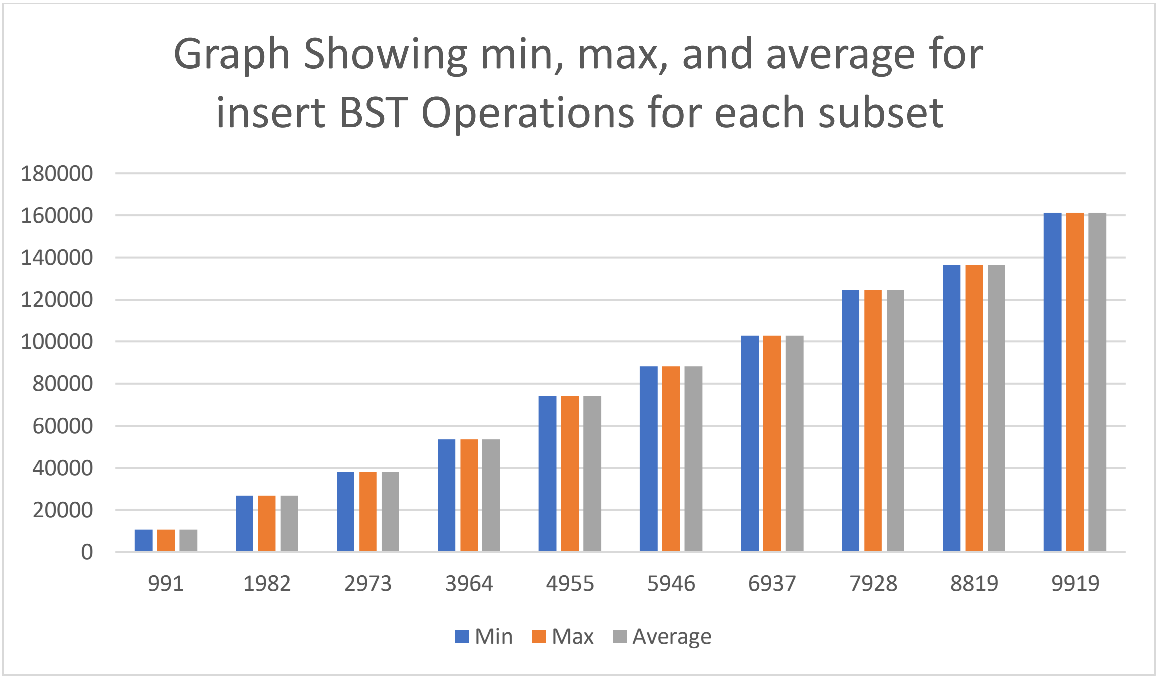
This set of input contained valid country names with an invalid date that is not included anywhere In vaccinations.csv and it produced output as expected, proving that VaccineArray and VaccineBST work with this type of input.

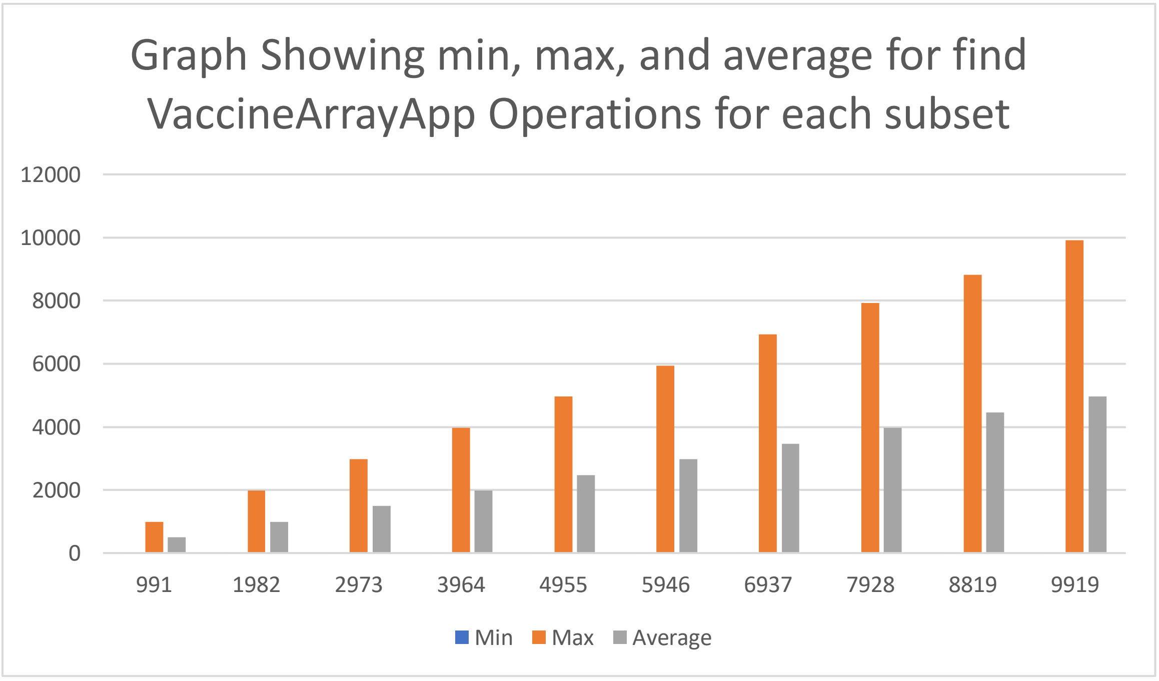
Thus, I was able to find that both apps work correctly when scrutinized with varying inputs.

Part 2 of Experimentation with results

To perform this part of experimentation, I first used the command "shuf -n [lines] [file] in the unix terminal to get each of the 10 subsets before I performed the experimentation on it each time. Next, I created a run.sh bash script that was used to run the VaccineArrayApp and VaccineBSTApp with input redirection from a file called textinput.txt that continuously contains one input and output redirection to an output.txt file. Lastly, I created a VaccineTest java program that iteratively writes to the aforementioned input file and then uses run.sh to run either app with the input file and to append the output to output.txt. I then used this output to calculate the min max and averages for opCount in each subset and recorded this. The graphs below represent the results.



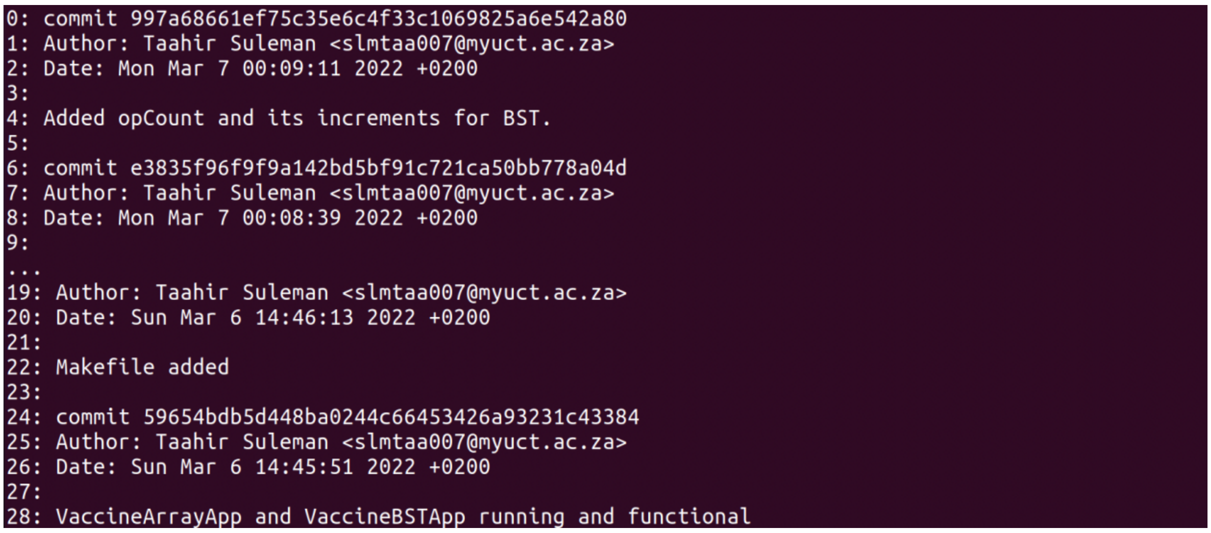




Explanation: Firstly, as portrayed by the second graph, the max, min and average case for the VaccineBSTApp insert operations are equal for each subset size since the same amount of inserts is performed each time the program is run and input is received. Furthermore, this equal value increases linearly with each 10% increase in subset size as more nodes are inserted with more data items present. Secondly, the find operations for VaccineBSTApp have the same min in each case being 1 since the best case scenario is finding the correct node at the root of the BST requiring one comparison. Furthermore, the max and average cases increase in the shape of a linear curve which matches theoretical expectations. Moreover, these maxes and averages increase with each increased subset size as more comparisons are being done per search. Thirdly, the min in each case for the VaccineArrayApp is 1 and is so small that it cannot sufficiently be shown in such a graph without creating an extremely large key: This matches theoretical expectations as the best case scenario is finding the object on the first comparison. Furthermore, the max and average cases increase in a linear fashion with each increase in subset size. Lastly, the difference in efficiency of the searching between the two apps can be depicted by the vertical scale in their respective graphs: the VaccineArrayApp graph scale increases in 2000's whereas the VaccineBSTApp's graph increases in 5's, a difference by a factor of 400, emphasizing that the VaccineBSTApp's search is much more efficient.

Creativity: The main area where I was creative was with the part 2 of experimentation where I created a java and bash script. I went over and above the assignment requirements and what we have learnt by learning how to write a bash script and make it executable in order to use one to run either VaccineArrayApp or VaccineBSTApp in each iteration of my experiment with both input and output redirection. I then used the output file to calculate what I required thus using my creativity to solve the second part of experimentation of this assignment. Furthermore, the assignment requirements recommended the usage of Unix or Python to perform this experimentation however I solved this part of the assignment primarily using a Java program, further highlighting the creativity present in my solution.

Git Log



```
0: commit 997a68661ef75c35e6c4f33c1069825a6e542a80
1: Author: Taahir Suleman <slmtaa007@myuct.ac.za>
2: Date: Mon Mar 7 00:09:11 2022 +0200
3:
4: Added opCount and its increments for BST.
5:
6: commit e3835f96f9f9a142bd5bf91c721ca50bb778a04d
7: Author: Taahir Suleman <slmtaa007@myuct.ac.za>
8: Date: Mon Mar 7 00:08:39 2022 +0200
9:
...
19: Author: Taahir Suleman <slmtaa007@myuct.ac.za>
20: Date: Sun Mar 6 14:46:13 2022 +0200
21:
22: Makefile added
23:
24: commit 59654bdb5d448ba0244c66453426a93231c43384
25: Author: Taahir Suleman <slmtaa007@myuct.ac.za>
26: Date: Sun Mar 6 14:45:51 2022 +0200
27:
28: VaccineArrayApp and VaccineBSTApp running and functional
```