CSC2001F: Assignment 1 report 2022

March

Authored by: Kevin Chiloane CHLKEV001

Binary Search Trees

Data structures

This assignment requires a written code to read a comma separated values file containing vaccination information vaccine(region,date,count) and store it in an array and a binary tree. Given the date and region, it should produce a count that matches the region.

Part 1 : Programs

For storing and retrieving data from the vaccinations.csv file, I created two applications-VaccineArrayApp and VaccineBSTApp. Each application contains methods that output the matching entries from vaccinations.csv.

VaccineArrayApp

Contain class VaccineArrayApp.

methods:

main method which contains the scanner to iterate over vaccination.cvs and store the entries.

Input reader to take user specified and input and store in array.

Use array of input to iterate the data array and includes conditions to read desired conclusion

-Start and stop to time the runtime of the program in milliseconds Run method to run the main program ---- writeToFile to save our output data to a text file

VaccineBSTApp

Contains two classess the main class VaccineBSTApp and NodeVaccineBSTApp class

Methods:

In the writeToile method, the number of operations is written using the FileWriter object.

- **The countFile function creates a file that stores the results of counting operations.**
- **Start and stop to time the runtime of the program in milliseconds Run method to run the main program.**
- **The printVaccine function takes a key which is the identity of the vaccine region as type String and outputs a specific data entry by vaccine region matched with vaccine count.**

In the main, read the text file and create a VaccineBSTApp object and call its methods.

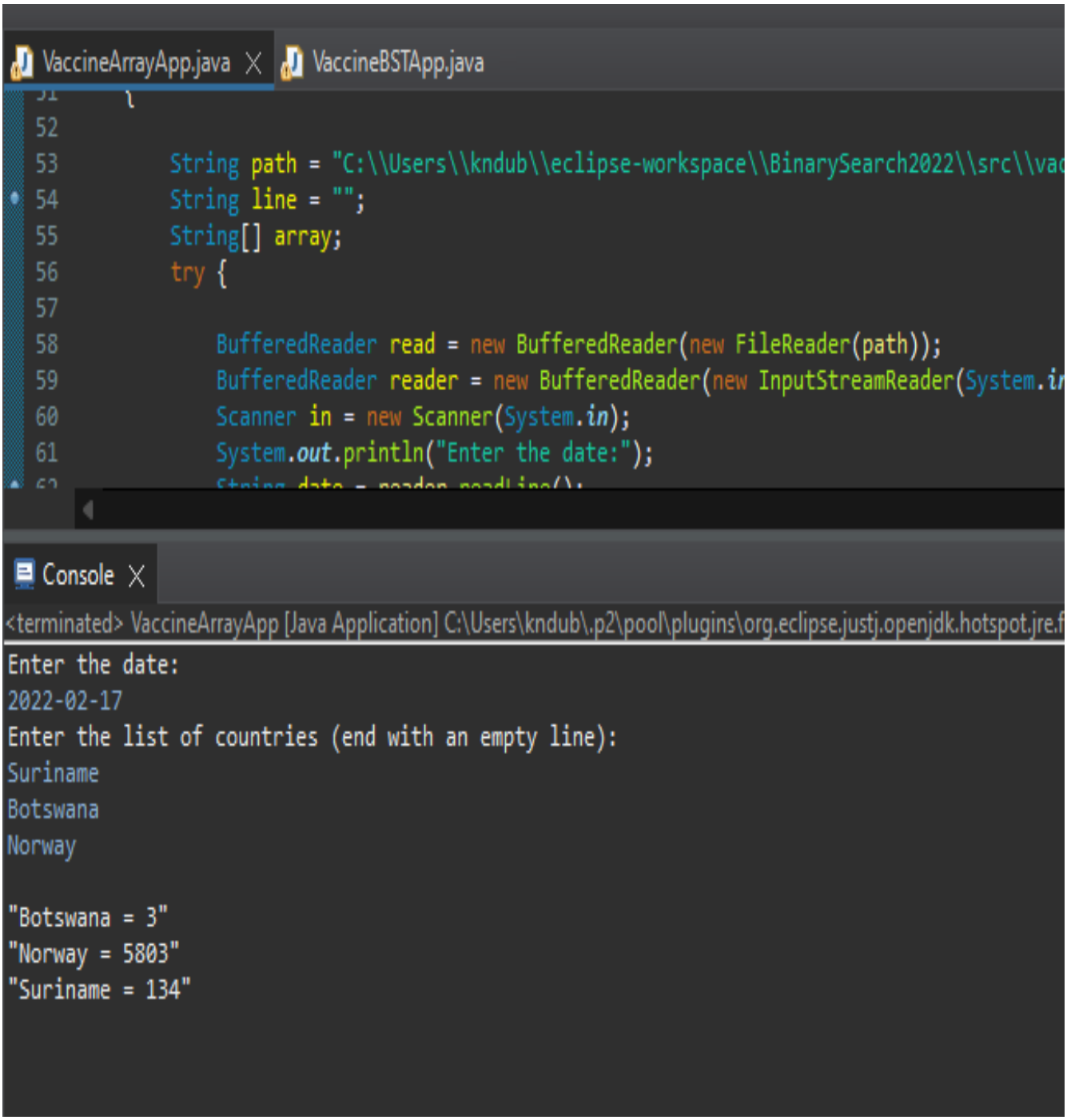
Class Node

Defines a constructor of Node to take Vaccine region and date. A to string method to printout desired outputs.

Part 2 : Experiment and testing

In order to determine which data structure is more efficient, we compared two data structures -- linear search and binary search -- based on their speed differences. VaccineArrayApp represents linear search and VaccineBSTApp represents binary search. Both contain arguments to count the number of operations and store the results.
Start and stop method at instrumental count and save the result to an output file
Testing each application with three valid arguments that work and three invalid arguments.
Following are snippets of the code running by testing values :

-VaccineArrayApp


VaccineArrayApp.java
VaccineBSTApp.java
{
String path = "C:\\Users\\kndub\\eclipse-workspace\\BinarySearch2022\\src\\vac
String line = "";
String[] array;
try {
BufferedReader read = new BufferedReader(new FileReader(path));
BufferedReader reader = new BufferedReader(new InputStreamReader(System.in
Scanner in = new Scanner(System.in);
System.out.println("Enter the date:");
Console
<terminated> VaccineArrayApp [Java Application] C:\Users\kndub\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.f
Enter the date:
2022-02-17
Enter the list of countries (end with an empty line):
Suriname
Botswana
Norway
"Botswana = 3"
"Norway = 5803"
"Suriname = 134"

-VaccineBSTApp

VaccineArrayApp.java VaccineBSTApp.java

```
74      {
75          if(currentNode != null)
76          {
77              printAll(currentNode.left);
78              System.out.println(currentNode);
79              printAll(currentNode.right);
80
81          }
82      }
83
84      public  Node printVaccine(String region,String a,String b)
85      {
86          countFile("data\\VaccineBStApp.txt");
87          Node currentNode = root;
88
89          while(!(currentNode == null))
90          {
91              if(currentNode.region.compareTo(region)> 0)
```

Console

<terminated> VaccineBSTApp [Java Application] C:\Users\kndub\.p2\pool\plugins\org.eclipse.justj.op

```
Enter the date:
2022-02-17
Enter the list of countries (end with an empty line):
Norway
Suriname
Botswana

"Botswana = 3"
"Norway = 5803"
"Suriname = 134"
```

Results

Table 1:

Data(n)	Name of Count	Average
500	VaccineArrayApp 500	10.985
1000	VaccineArrayApp 1000	11.120
1500	VaccineArrayApp 1500	11.263
2000	VaccineArrayApp 2000	11.389
2500	VaccineArrayApp 2500	11.523

Table 2:

Data(n)	Name of Count	Average
500	VaccineBSTApp 500	09.344
1000	VaccineBSTApp 1000	10.088
1500	VaccineBSTApp 1500	10.355
2000	VaccineBSTApp 2000	10.648
2500	VaccineBSTApp 2500	10.825

Discussion of results:

Average case0(n)time when the required match is at the middle of list and worst 0(n)time case when the required is at end of the list.

The VaccineBSTApp all operation run in 0(logn) time.

In comparison to the linear search algorithm, the binary search tree algorithm is more time-efficient as it can run all operations with a constant fast time complexity, while the linear search algorithm's best case is slightly slower than the binary search algorithm's worst case.

VaccineBSTApp spends more time balancing the tree during data insertion, which impacts its performance. The Vaccine data structure is also more efficient at searching information because the tree has already been balanced.

By using the VaccineBSTapp, the desired match was located at the beginning of the balanced tree by just two operation counts, which is the best case.

The VaccineBSTApp it was able to locate the desired match which was at the middle of the balanced tree by 9 operation counts which is the average case: 0(logn). The VaccineBSTApp it was able to locate the desired match which was at the end of the tree by 11 operation counts which is the worstcase: 0(logn).

Creativity:

The use of start and stop will allow the instrumental count Python script to calculate the number of steps the program takes to split the data into 500 differences and locate matching values.