9/3/2022

Report

Data structure (BS-tree)

Tokelo Makoloane–MKLTOK002

COMPUTER SCIENCE

Problem Description

Given a file containing a list of entries having multiple fields (County, Date , Vaccination number) that has not been processed for use (raw data), It is required to use this data to create an APP (which uses a Binary Search Tree as an internal data structure) that has the several functionalities and use a traditional array to perform the same operations and compare which one is more faster to search the data for respective vaccine numbers.

Application design

In this application we going to design two Apps that will read in the given data file that contains the country, date and the vaccine number.

The first application designed is the VaccineArayApp , which uses a traditional array to search through the given file and check for which country matches with the corresponding date.

The second application design is the VaccineBSTApp, which uses a Binary search tree.

1.VaccineArrayApp

Only contains the main method that uses a 2D array to store data. What it does it will create an array fist that will store the input and that array will be used to compare the country of the fist entry of all the arrays in the outer array and If a certain country exist is the iterated arrays then check if the corresponding date matches the inputted date. If everything matches then it will yield the expected vaccine number.

2.VaccineBSTApp

- **CLASSES USED : ,**
    - **BinaryTreeNode ,**
    - **VaccineBSTAPP : main()** method only
    - **FileBSTApP:**
        - **METHODS:**
            - **void** Readfile(String)
                - This method read the file and store the data in a binary search tree as form of String.
            - **BinaryTreeNode** find(String)
                - This method will find the corresponding country and date and output the respective vaccine numbers.
            - **Boolean** ifNodeExists(BinaryTreeNode, String)
                - This method will check if the node found by the ‘find method’ where it exists in the tree checking all the branches recursively.
            - **void** insert(String)
                - This method will insert the lines of the files recursively in a search tree.

TEST VALUES AND THEIR OUTPUT.

1st trial:

Test for 2022-02-17 date:

Countries:

1. **Israel = 4346**
2. **Egypt = 0**
3. **Sierra Leone = 0**
4. **Dominica Republic = 17751**
5. **South Namibia = <Not Found>**

2nd trial:

Test for 2022-01-17 date:

Countries:

1. **Tokelo = <Not Found>**
2. **North Korea = <Not Found>**
3. **Lesotho = <Not Found>**

3rd trail:

Test for 2022-01-09 date:

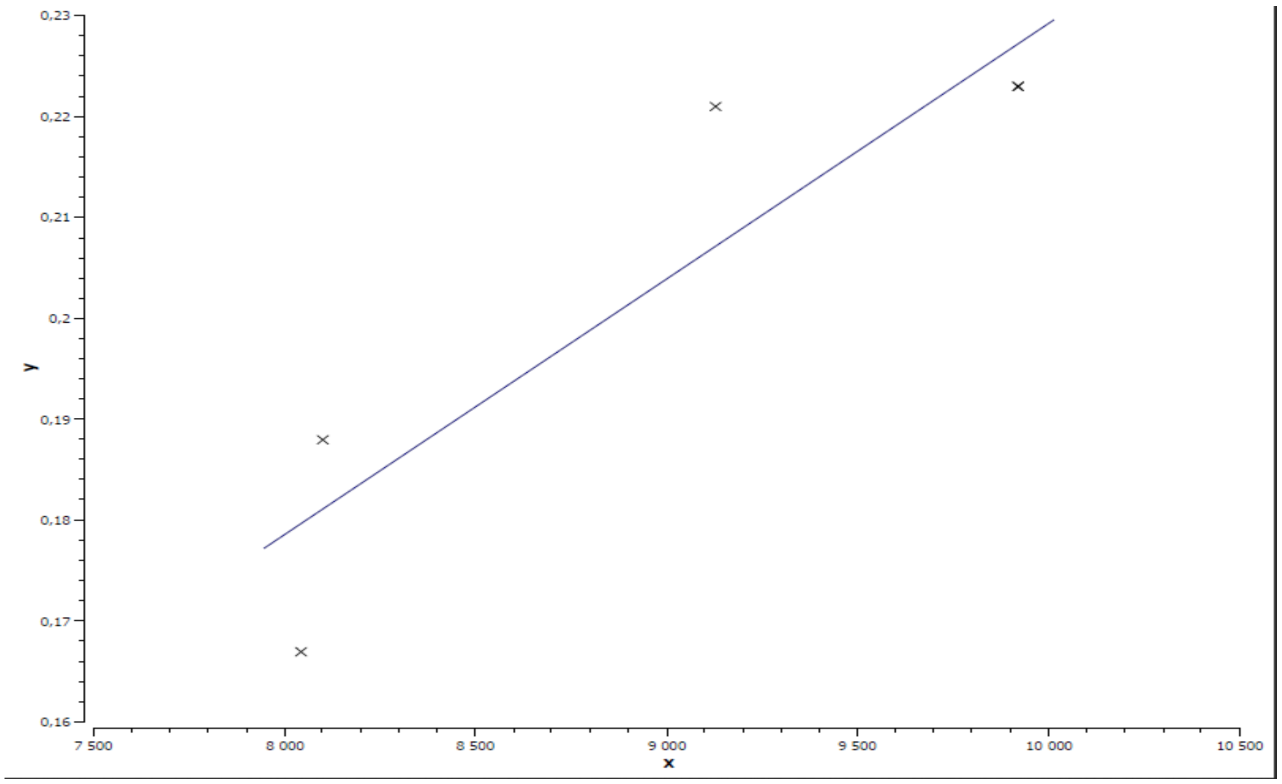
Countries:

1. **Barbados = 152**
2. **Cape Town = <Not Found>**

Tables and graphs

1. **For Array**

	OPERATIONS	**TIME(S)**
1st trial	1. **For ISREAL => 9919**	**0.223**
	2. **For Egypt => 9129**	**0.221**
	3. **For Sierra Leone => 8042**	**0.167**
	4. **Dominica Republic => 8099**	**0.188**
	5. **South Namibia => 9919**	**0.223**



- **For the Array operations the results of the trials gave the expected complexity that will yield a linear correlation, and since the points are a-bit away from the regression line it shows there more a country is searched in the array and it happens is in the last index of the array the more and more the operations be closer the size of the array which is 9919 elements in an array. So the time complexity should be O(n) since the APP will search every element of the array. The worse case is when the element is at the end of the array, which means it must check all the elements until it reach the end of the array**

2. **For Binary Search Tree.**

	OPERATIONS	**TIME(S)**
1st trial	1. **For ISREAL => 21**	**0.0120**
	2. **For Egypt => 24**	**0.13**
	3. **For Sierra Leone => 24**	**0.17**
	4. **Dominican Republic => 16**	**0.11**
	5. **South Namibia => 21**	**0.087**

- **According to the above table the results are a-bit closer to each other and the time it took to complete an operation for each case they are close to each other varying with a small amount of time between them. Which shows the time complexity on an average case is O(log n) , since when it happens the element that is searched for Is in the last node of the tree, it doesn't search all the nodes of the tree but search mostly the roots of the subtrees.**