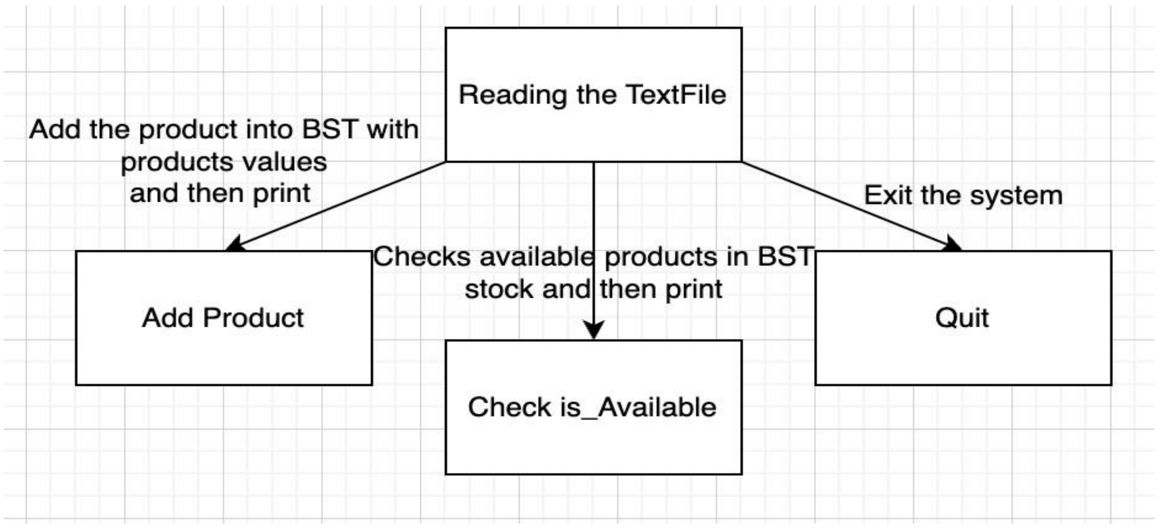
PROGRAMMING ASSIGNMENT 3

Question 1

Problem Statement and Code Design

For the first question of this programming assignment, we aimed to create an Inventory Tracking System using search trees. One of the efficient implementations of building this inventory tracking system would be the use of a binary search tree (BST) data structure. According to reach that goal, we have used 3 basic operations. Additionally, the code includes several sub-parts intended for managing compatible designs. These sub-modules are basically defined utilizing a structure chart below.


Reading the TextFile
Add the product into BST with products values and then print
Checks available products in BST stock and then print
Exit the system
Add Product
Check is_Available
Quit

Implementation & Functionality

As we mentioned above, our implementation contains 3 main operations which they are: addProduct, isAvailable and Quit. In a Binary Search Tree, each node contains a key and two children nodes, which are the left and right subtrees of the current node. When inserting a new product into the tree, we can compare the product's ID to the key of the current node and decide whether to insert the product into the left or right subtree. If the product's ID is equal to the key of the current node, we can update the product's information stored at that node.

First of all , since we need to have the input elements, we wrote the Main Method as contains BufferReader for taking the input.txt file. We thought accessing the element of the text file separately is suitable and efficient solution for this problem. Therefore, we used trim and split method to breaks the given string and returns a string array.

AddProduct: To implement the create product operation, we can define a Product class that contains the ID, name, and piece attributes of the product. Then we can define an insert method that takes a Product object as an argument and inserts it into the BST.

CheckisAvailable: Moreover, to build the product available operation, we can define a method that checks stock accessibility recursively. It takes a product ID as an argument and searches for the product with the key in the Binary Search Tree. If the root node is null, then it means that the system does not include a product with searched id. On the other statement, if the product is found, the method will print the number of pieces available for that product.

Otherwise, it will print a string message which indicates the product is out of stock. In this way, we provide a searching method that does not require visiting every single node.

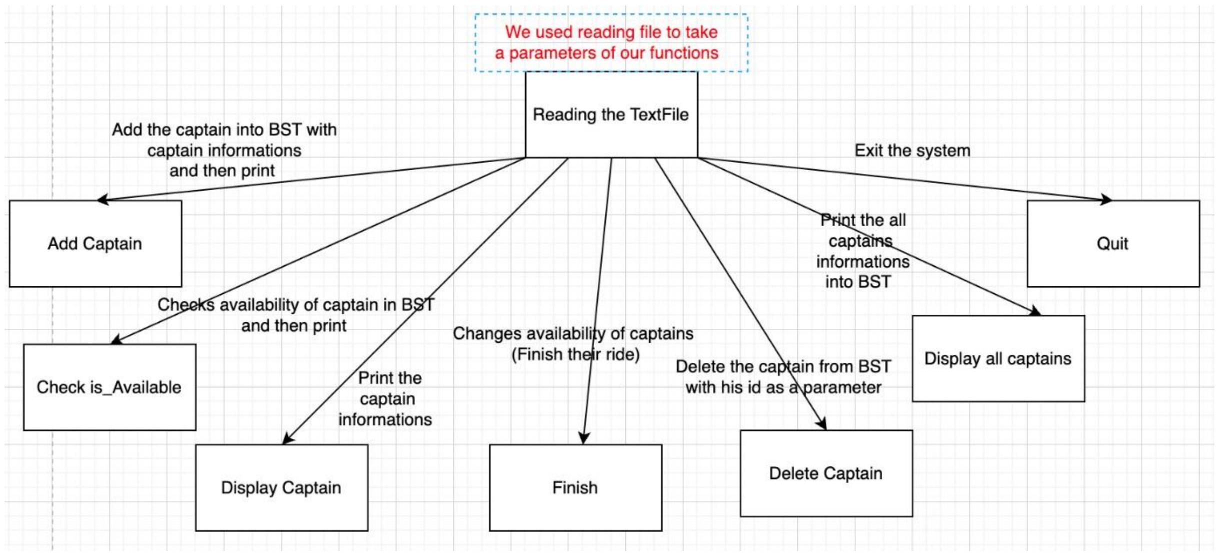
Quit: It can be implemented by defining a method that simply stops the program.

Finally, to avoid any unexpected outcomes we consider the exceptional situations and designed our code according to them.

Question 2

Problem Statement and Code Design

For the second part of the question, we, created a chauffeur-driven rental company (CDRC) which used for managing system code using binary search trees. CDRC provides a system that we can control the captain's salary increase. We have a database for chauffeurs which should kept in the management system. In this database we should define some attributes which they are: ID, name, available information and the rating star. Also, the chauffeur-driven rental company need to have 7 sub- functionalities which written to make the code adaptable which listed above.


We used reading file to take a parameters of our functions
Reading the TextFile
Add the captain into BST with captain informations and then print
Exit the system
Add Captain
Quit
Print the all captains informations into BST
Checks availability of captain in BST and then print
Changes availability of captains (Finish their ride)
Display all captains
Delete the captain from BST with his id as a parameter
Check is_Available
Print the captain informations
Display Captain
Finish
Delete Captain

Implementation & Functionality

In this second question, we have 7 functionalities listed as following: add Captain, delete the captain, print captain's information, print all captains information, rent a car, finish the ride and quit.

Firstly, in order to use the information on the captain database we used a reader to take the inputs. With BufferReader method we taking the needed elements separately in an efficient way.

insertCaptain: To implement the insertCaptain method we need to keep two values which are: the captain ID and the captain's name. For this function, since in the inserting operation requires adding a new captain, there will be no chance to availability of this captain is false.

Therefore, the default value for the rating star set to 0 and the default value of available is true.

deleteCaptain: In this method, we need to take one value which represents the captain's ID. For the delete command, we search the id to be deleted in the tree, when it is equal to the node ,then remove the node of specific captain from the BST. When we finish this operation, it will print the message which indicates captain left CDRC. If the specified captain is not found, it will output a message which shows there isn't any captain with that id number.

DisplayCaptain: The display captain method accepts two parameters: root node and id. It checks if there is a captain with an input id, then prints the captain's information.

DisplayAllCaptains: The Display All Captains method takes a parameter, which is the root node, then traversal logic, visits all nodes (captains), and prints their information.

CheckisAvailable: This method checks the condition of availability of captains in a recursive way. It takes the captain's id as a parameter and compares it with a key to search, then if the statement which is checking availability is equal to 0 or not then prints availability.

Finish: The root node and an int key are parameters for the finish method. There are if statements in the method body. Because the key is unique, we can use it to check Captain using his ID. If there is a captain with the entered ID, then it checks whether he is riding or not; if he is, it makes him available, and the conditional rating star points can increase. Lastly, after the captain's availability changes, it also prints the captain's information. **Quit:** It can be implemented by defining a method that simply stops the program.

Testing

At first, it was a little difficult to pull the necessary input from the text file. because the text file contained both integer values and string values. There were also spaces between the lines. But then we pulled the information from all rows with the logic of the string array. For those with int values, we used the parseInt() method. We constructed the spaces in our output completely and actually passed all test cases in accordance with the expected output in the VPL and the homework pdf.

FINAL ASSESSTMENTS

1. Creating a comprehensive test class was one of the most difficult components. Because we didn't have access to any other library or utility, I attempted to stick to my own simple test class implementation rather than utilizing a production-ready test framework. The second and most difficult aspect was creating such a superb algorithm to determine the most often occurring three keys.
2. After completing this project, we noticed that the BST logic fits quite nicely. We have observed the structure of the BST approaches.
3. We were able to write more easily, especially since we had previously completed tasks where we learnt how to extract information from a txt file.