DETAIL DESIGN – PS2 (VEHICLE RECORDS)

PROBLEM STATEMENT : A warehouse has certain number of trucks that transport supplies in and out of the warehouse. Each truck has a unique identifier and a counter to keep a track of how many orders did each truck fulfilled. Whenever a truck moves in/out of the warehouse its unique ID is recorded. Every day when the truck arrives at the warehouse for the first time, the counter is set to 0.

From then onwards, the counter is incremented each time the truck enters and exits the warehouse. When the truck will leave the warehouse to deliver supplies to the client, an order will be opened against its ID in the system and the counter will be 'odd'. This is considered as an open order. The order will be marked as closed when the truck arrives back at the warehouse and the truck's counter will be 'even' at that time. If the counter is 0, it means they have not started working and have just arrived to the warehouse for the first time in the day. Hence the counter indicates the number and status of orders.

The warehouse manager requires assistance in keeping a track of open and closed orders through an automated system. In addition to this each truck can do a maximum of 'x' deliveries in a day. The implemented system will also help the manager in assigning orders to each vehicle. It can identify which trucks have completed their maximum deliveries for the day and which ones can still be assigned with more orders.

The warehouse manager uses the above system to answer the below questions:

1. Total vehicles that came to the warehouse for work?
2. Check specific truck whereabouts
3. Number of open, closed and yet to be fulfilled orders
4. List of trucks that have moved in/out of the warehouse more than 'z' number of times.
5. List and number of trucks that have completed their maximum deliveries for the day
6. List and number of trucks that are currently in the warehouse and available to deliver supplies

1. Purpose and Scope

This document describes the Low-level design for the Vehicle records system. It aims to provide the detailed technical descriptions.

2. Solution Overview

The vehicle records system is maintaining the records of the vehicle(trucks) which are being used at a warehouse. It has multiple functionalities which gives us the truck status, trucks availability, high frequency trucks and order status etc. It also provides a feature of updating new truck entries in the system.

3. General Assumptions

1. The program only accepts the inputs in the specified format as given in the requirement.

2. Exception handling is done for the data structure used in only inserting and deleting a new record.
3. Input and output filenames are hardcoded as per the requirement given. No other input files other than mentioned in the requirement will be considered.
4. The input files are mandatory for the program to execute.

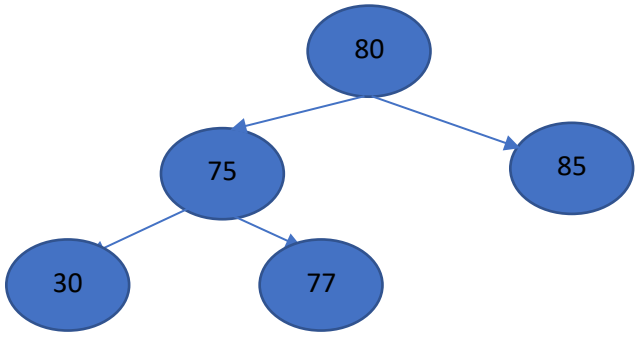
4. Data Structure Model – Binary Search Tree ADT

The abstract data type binary tree is being used to implement the data structure/storage. It gives an option to recursively traverse the tree structure. The insert and update basic operations are implemented explicitly instead of using any library functions in Python.

The BST is allocating the data dynamically and performance wise its better than linked list as it's not linear. Unlike linear data structure which have only one logical way to traverse them, It can be traversed in different ways. The algorithm is written in such a way that traversing in ascending order (as per the requirement) is simpler. The in-order traversal (left, root, right) is used to display the data. The storage is in such a way that left node is always smaller than the root and right node is always bigger than the root.

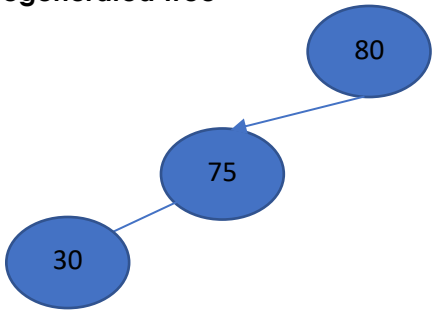
Each BST node is holding its own value and counter and the address of its left and right child (if either or both the child is present). The input file is being read and stored in the BST. For any new node addition, a binary traversal is being done and attached to the relevant location. The first node entry is always become the root. The subsequent entry either go to the right or left depending upon the node value.

Example (BST):



There can be a **worst-case complexity** scenario of a BST, when a binary search tree is fully degenerated, and it becomes like a chain of n nodes. It eventually become a linked list and its height becomes O(n), thus making the search complexity as O(n).

Example: **Degenerated Tree**



5. User APIs

5.1. **readTruckRec** - Reads vehicle ids entering and leaving the warehouse from the inputPS2.txt file.

In this API, the BST is built for the input given. The truck nodes are stored as per the BST storage policy and the truck counter is updated as per the entries in the input. The function is called recursively from the root node.

The first entry in the inputPS2.txt is stored outside the BST as a global max limit allowed for any truck to fulfill the order.

Worst case time complexity = O(n)

Average case time Complexity = O(logn)

5.2. **updateTruckRec** - Updates the existing system with the truck ID entering and leaving the warehouse from the promptsPS2.txt file

In this API, depending upon the input, either a new node is created or exiting node counter is increased in BST. There is a check that, if max fulfillment limit is reached for any truck Id, the counter will not be updated, and a message will be given.

Worst case time complexity = O(n)

Average case time Complexity = O(logn)

5.3. **printTruckRec** - Counts the total number of trucks that came to the warehouse for work and prints the list of vehicle ids added into the system and their counter separated by a ',' in outputPS2.txt (as per the requirement given)

In this API, the BST is being traversed in in-order recursively and entries stored in a list. Once the traversal is completed, it print the total truck count and truck ids with their counter in specified format.

e.g.

Total number of vehicles entered in the warehouse: 03

31, 1

34, 4

56, 0

Worst case time complexity = O(n)

Average case time Complexity = O(n)

5.4. **checkTruckRec** - Reads the truck id from the promptsPS2.txt file to be searched for availability in the system and print the status accordingly in outputPS2.txt file (as per the requirement given)

In this API, the BST is being traversed in in-order recursively and searched for the truck Id. If the entry is found, a relevant message is printed based on the counter of the truck Id. If the entry is not found, the message will be given as not vehicle doesn't entered the warehouse today.

Example:

For counter as odd (5):

Vehicle id 55 entered 5 times into the system. It is currently fulfilling an open order

For counter as even (4):

Vehicle id 65 entered 4 times into the system. It just completed an order

For counter as 0:

Vehicle id 75 just reached the warehouse

For entry not found:

Vehicle id 771 did not come to the warehouse today

Worst case time complexity = O(n)

Average case time Complexity = O(logn)

5.5. **printOrderStatus** - Prints the number of open, closed and yet to be fulfilled orders out of the total target orders into the outputPS2.txt file

In this API, the BST is being traversed in in-order recursively and calculated for the sum of total orders handled. It takes the input as total number of order and prints the order status accordingly in outputPS2.txt file.

The odd counters are for total open orders. The even counters are for total closed orders. The remaining are the 'yet to be fulfilled'.

Note: 'yet to be fulfilled' can't be less than 0.

e.g. The following status of 11 orders:

Open Orders: 2

Closed Orders: 2

Yet to be fulfilled: 7

Worst case time complexity = O(n)

Average case time Complexity = O(n)

5.6. **highFreqTrucks** - Generates the list of trucks that have moved in/out of the warehouse more than a given number (input) of times and prints this into outputPS2.txt

In this API, the BST is being traversed in in-order recursively and the counter of each truck with its id is stored in a list. Once the BST traverse is completed, the list is being checked for count greater than or equal to the given number of high frequency and printed in outputPS2.txt file. If not qualified any entry, a specific message is printed in the output.

Worst case time complexity = O(n)

Average case time Complexity = O(n)

5.7. **maxDeliveries** - Prints the count and list of trucks ids that have completed their maximum deliveries for the day.

In this API, the BST is being traversed in in-order recursively and counter/2 is checked against the delivery limit. If it satisfies, it's appended into the list. Once the BST traverse is completed, the list is printed in outputPS2.txt file for max delivery and truck ids. If not qualified any entry, the message is printed in the output with count 0.

Worst case time complexity = O(n)

Average case time Complexity = O(n)

5.8. **availTrucks** - Prints the count and list of vehicle ids that are currently in the warehouse and available to deliver supplies.

In this API, the BST is being traversed in in-order recursively and counter%2 is checked against 0. If it satisfies, the counter is being checked against max limit. If its less than max limit, its appended into a list. Once the BST traverse is completed, the list is being printed in outputPS2.txt with count of trucks available and its Ids.

Worst case time complexity = O(n)

Average case time Complexity = O(n)

6. Alternate way of modelling the problem

The other option which could have been used to model the data structure and relevant operations is through a linked list. However, the average case performance of a BST is better than linked list, as explained below in the table.

Execution	Data Structure	Access	Search
Average Case	Linked List	O(n)	O(n)
	BST	O(logn)	O(logn)
Worst Case	Linked List	O(n)	O(n)
	BST	O(n)	O(n)

The average case of BST in case of access and search is better compared to the linked list execution.