BSc (Hons) in Computing

Level 5

INDIVIDUAL ASSIGNMENT

Module Code & Title:

COMP50004 – Databases and Data Structures- 2

Prepared By:

Student Number	Student Name	Cohort
CB010324_ 21035776-1	Kahanda Gamage Yasiru Udana	IF2321SE

Date of Submission: 25. 09. 2023

Instructor: Mr. Guhanathan Poravi

MARKING CRITERIA	%	MARKS OBTAINED
TOTAL (%)		

Table of Contents

Problem 1: Optimizing the loading of parcels tot the delivery van

1. Introduction

Quick Box, a well-known delivery firm established in Sri Lanka's rich and diverse geography, has long been at the forefront of the country's logistics industry, always aiming to exceed client expectations. Quick Box, every forward-thinking company, faces operational problems. In this study, It address a major issue that Quick Box has experienced in its daily operations: effective parcel loading into delivery trucks.

By methodically defining a comprehensive solution, this study serves as a model for tackling the package loading issue. The strategy will center on establishing the necessary data structures and outlining the data that must be stored inside these structures. Following that, It will dive into the practical parts of developing this solution, which will include generating sample data indicative of Quick Box's activities and establishing a thorough set of functions that will serve as the foundation of our solution.

This report will also provide a pseudocode representation of the suggested solution to guarantee clarity and practicality. While pseudocode provides an abstract representation of the solution's logic, it's vital to understand its limits, which will go over later. In addition, report will look at potential future additions that might improve Quick Box's efficiency and service quality.

Finally, it is critical to emphasize that this study is built on some fundamental assumptions, which will be fully explained. These assumptions serve as the foundation for our suggested solution, providing for a clear grasp of its viability and application within the context of Quick Box's operations. In the report, will analyze, create, and optimize the package loading process, with the ultimate goal of improving Quick Box's operational excellence and customer pleasure.

2. Data Structures used.

In Java, a data structure is a set of data pieces that are used to store and organize data so that it may be utilized more efficiently. The data structure used for this problem is arrays, LinkedList, and stack. Arrays are most basic data structure in java. An array is a linear data structure, known as a "list," in which elements are stored. With the help of the index, can randomly access elements in this data structure. Linked list is a linear data structure. It is made up of connected nodes. A linked list is a collection of nodes that are kept in memory at random. A node has two parts. They are data part and address part. Likewise, all the other selected data structure stack also a linear data structure. It uses the last in first approach. It has only one end.

Relevant data for selected data structures.

Arrays: customer Id, customer name, delivery location, Route, order Id, height of the parcel, length of the parcel, width of the parcel, delivery date.

LinkedList: Details of the orders which will be sent for the van for the loading.

Stack: To create the loading list of goods based. Because last in Item should be delivered first.

Reasons to select above data structures.

Arrays: easy sorting, replacement of multiple variables, easy iteration.

LinkedList: capacity and size always equal, dynamic in size, and suitable for instances where require deletions and insertions.

Stack: follows Last in, first out approach, have 4 operations, and only top element is able to access.

3. Sample Data for Loading order

parcel_id: P001,
customer_name: John Smith
delivery_address: 456 Elm St, City A
description: Fridge,
route: A

parcel_id: P002,
customer_name: Jane Doe
delivery_address: 789 Oak St, City C
description: Statue
route: C

parcel_id: P003,
customer_name: David Brown
delivery_address: 79 Pine St, City C
description: Kids toy
route: C

parcel_id: P004,
customer_name: Sarah Green
delivery_address: 33 Maple St, City B
description: TV
route: B

van_capacity:
heigth: 150 cm
width: 100 cm
length: 340 cm

parcel_size:
heigth: 100 cm
width: 60 cm
length: 50 cm

route:
A: 5 km
B: 10 km
C: 20 km

4. Sample Data for Invoice

invoice_id: INV001
customer_id: CUST001
customer_name: John Smith
customer_address: 456 Elm St, City A
route: A
invoice_date: 2023-09-15
total_amount": 65.00

parcels": [
  parcel_id: P001
  description: Fridge
  quantity: 1
]

invoice_id: INV002
customer_id: CUST002
customer_name: Jane Doe
customer_address: 789 Oak St, City C
route: C
invoice_date: 2023-09-19
total_amount": 55.00

parcels": [
  parcel_id: P002
  description: Statue
  quantity: 1
]

invoice_id: INV003
customer_id: CUST003
customer_name: David Brown
customer_address: 79 Pine St, City C
route: C
invoice_date: 2023-09-12
total_amount": 75.00

parcels": [
  parcel_id: P003
  description: Kids Toy
  quantity: 1
]

5. Functionalities

To solve the problem related with the loading the parcels to delivery van involve a number of functionalities and features, to improve the loading process more effective and accurate. Below listed are the functionalities that would need to be included in the solution,

- **Managing Space** - Consider the weights and size of the parcel to distribute the load evenly in the van.
- **Route Optimizing**- Parcels are loaded in a way that they will delivered to minimize the time and distance.
- **Loading goods to van**- able to add
- **Error Handling**- Provide warnings when incomplete data is added or when overloading the van.
- **Historical Data Storage**- Maintain past data of the loading process to get reference in the future.
- **Generate invoices**- Create invoices to including all the necessary details of the delivery to give respective customer.
- **Generate reports**- Get reports of the loaded parcels for make decisions.
- **Generate loading parcels**- Get details and order of the parcels that need to be loaded to the van.
- **Retrieval of details**- Retrieve details of the customer and delivery quickly for the decision making.
- **Storing details**- Able to store and organize data of the delivery and customer in appropriate data structure for the effectiveness and accuracy.

6. Pseudo code

```
Structure Package:
    Integer distance  //The distance of the package delivery
    String details   //Details about the package

   //Constructor to initialize a package
    Constructor Package(distance, details):
        Set this.distance to distance
        Set this.details to details
End Structure


// Define a function to perform merge sort on a list of packages
Function MergeSort(packages)
    Integer size = Size of packages
    If size <= 1
        Return packages

    // Split the list into two halves
    LinkedList left = Initialize LinkedList
```

```
    LinkedList right = Initialize LinkedList
    Integer middle = size / 2
    For i from 0 to middle - 1
        Add packages[0] to left
        Remove packages[0] from packages
    End For
    While packages is not empty
        Add packages[0] to right
        Remove packages[0] from packages
    End While

    // Recursively sort both halves
    left = MergeSort(left)
    right = MergeSort(right)

    // Merge the sorted halves
    Return Merge(left, right)
End Function

// Define a function to merge two sorted packages lists
Function Merge(left, right)
    LinkedList result = Initialize LinkedList

    While left is not empty and right is not empty
        If left[0].distance >= right[0].distance
            Add left[0] to result
            Remove left[0] from left
        Else
            Add right[0] to result
            Remove right[0] from right
        End If
    End While

    // Append remaining packages from left or right if any
    While left is not empty
        Add left[0] to result
        Remove left[0] from left
    End While
    While right is not empty
        Add right[0] to result
        Remove right[0] from right
    End While

    Return result
End Function

// Define a function to load packages onto the delivery van
Function LoadPackages(packages)
    LinkedList sortedPackages = MergeSort(packages)
    Stack deliveryStack = Initialize Stack
```

```
    For each package in sortedPackages
        Push package onto deliveryStack
    End For

    While deliveryStack is not empty
        Package currentPackage = Pop from deliveryStack
        ProcessPackage(currentPackage)
    End While
End Function

// Define a function to process a package (replace with actual logic)
Function ProcessPackage(currentPackage)
    Print "Loading package: " + currentPackage.details
End Function

// Example usage
LinkedList packages = Initialize LinkedList
Add new Package(10, "Package 1") to packages
Add new Package(5, "Package 2") to packages
Add new Package(15, "Package 3") to packages

LoadPackages(packages)
```

7. Limitations of Pseudo code

- Lack of Error Handling: There are no mechanisms for handling errors in the code. For instance, it cannot manage situations in which the distance between packages is the same or if the inputs are incorrect.

- No handling of vehicle capacity: The delivery van's capacity is not taken into account by the code. It's assumed that the vehicle can fit all of the packages. You would need to make allowances for the van's potential capacity since it may have a finite amount of space.

- The method makes the assumption that you can load and sort all packages into memory at once. Due to memory limitations, in a real-world scenario with several packages, this might not be possible.Processing of packages is not complete since the ProcessPackage method only serves as a placeholder and does not contain any processing logic. Processing packages could need more complicated operations, depending on the real requirements.

- Limited Flexibility: The code is designed only for distance-based package sorting. The sorting algorithm would need to be changed if you wanted to organize packages according to different standards or use more intricate methods.

- The code assumes that packages are sorted once and loaded into the van, therefore there is no consideration for real-time updates. The code does not address how to handle such in-

the-moment modifications when package information can actually change dynamically (for example, when new packages are added or package statuses are altered).

- Assumption of a Simple Package Structure: The distance and details are the only components of the package structure. In reality, packages might have more attributes, in which case you would need to modify the data structure.

- No Reporting or Output: The code doesn't offer a way to output or display the order in which products are loaded onto delivery vans.

- The code is dependent on the platform and language pseudo code, a high-level representation. You would need to modify it to work with a particular programming language and platform, each of which could have restrictions and needs of its own.

- Merge Sort is an efficient sorting method, but its complexity can have an impact on how well it performs on huge datasets. Depending on the length of the package list, you can look at various sorting methods for improved performance.

8. Functionalities to include in the future.

- **Parcel Sorting**- Sorting according to customer, location, loading order, urgency.
- **Multiple Delivery vans**- Managing loading orders for multiple vans.
- **Different size of parcels**- get in parcels from different sizes and managing the loading order.
- **Scalability**- Creating a system that can handle an increasing number of parcels and delivery routes.
- **Constrains handling**- Handling vehicle capacity, priority delivery, fragile parcels.
- **Return Management-** Handling return parcels effectively.
- **Integrating with Inventory Management-** To ensure tracking of the parcel.

9. Assumptions made for first scenario.

The assumptions made to solve the issue with the loading parcels to van faced by QuickBox.

- Single van been used for the loading process.
- Only one door is used for the loading of the goods. The door used for the loading is the located at the back of the van.
- There are no errors in the details of the parcel.
- All the parcels are in uniform size and weights.
- There are no delivery priories for the goods.
- Dimensions of the van will be height-150cm , width- 100cm, length- 340cm.
- There is a capacity constraint, cannot be filled more than the capacity of the van.

- One destination per parcel.
- Dimensions of the parcel will be height 100cm, width 60cm, length 50cm.
- All the parcels are handled equally.

Problem 2: Optimizing the Delivering of parcels to the customers.

10.Introduction

QuickBox is now faced with optimizing its van delivery schedule in response to the successful resolution of parcel loading difficulties. This study provides a comprehensive solution by outlining important data structures and the relevant data to be kept inside them. Following that, the paper goes into the generation of sample data and an enumeration of critical solution functions. It also includes pseudo code for implementation, addresses inherent limits, and suggests potential future upgrades. Furthermore, the paper emphasizes significant assumptions made during the development of the solution. Distribute packages among cars to provide a balanced workload while reducing the number of vehicles required. Data Structures used.

To find a solution for the delivery route issue, there are some select data structures we used, they are arrays, specially the 2D arrays, linked List, stack. Advantages of using data structures are efficient processing, it has powerful data searching capabilities, and enhancing application performance.

Relevant data for selected data structures.

Arrays: customer Id, customer name, delivery location, Route, order Id, height of the parcel, length of the parcel, width of the parcel, delivery date, delivery location coordinates, shortest route.

2D Arrays: The distance between each delivery location.

LinkedList: Details of the orders which will be deliver.

Stack: To create the unloading list of goods based on the delivery locations. Because last in Item should be delivered first.

Reasons to select above data structures.

Arrays: Random access, Easy sorting, replacement of multiple variables.

2D Arrays: Distances between cities are simpler to visualize in a matrix style than in a flat list.

LinkedList: Capacity and size always equal, dynamic in size, and Efficient memory allocation.

Stack: follows Last in, first out approach, All the four operations are of O(1) complexity, and linear data structure.

11.Sample Data for delivery route of the goods

Parcels

parcel_id: P001
destination_address: 123 Main St, City A

deadline: 10:00 AM

parcel_id: P2,
destination_address: 456 Elm St, City B
deadline: 11:30 AM,

parcel_id: P3
destination_address: 789 Oak St, City C
deadline: 12:30 PM

Delivery Locations

location_id: L1
customer_name: Customer A
address: 123 Main St, City A
time_window: 9:00 AM - 1:00 PM
latitude: 12.345
longitude": -45.678

location_id: L2
customer_name: Customer B
address: 456 Elm St, City B
time_window: 10:30 AM - 2:00 PM
latitude": 23.456,\
longitude": -34.567

depot:
location_id: Depot
name: Depot
address: 100 Warehouse Rd, Depot
latitude: 0.000
longitude: 0.000

Vehicles

vehicle_id: V1
capacity: 100x150x340 (wxhxl)
type: Van
starting_location: 100 Warehouse Rd, Depot

12.Sample Data list of grid co-ordinates

Backup Data

latitude: 40.7128

longitude: -74.0060
location_name: 456 Elm St, City A

latitude: 34.0522
longitude: -118.2437
location_name: 789 Oak St, City C

latitude: 51.5074
longitude: -0.1278
location_name: 79 Pine St, City C

latitude: 48.8566
longitude: 2.3522
location_name: 33 Maple St, City B

latitude: 0.000
longitude: 0.000
location_name: Depot

Goods

item_id: Item001
description: Fridge
quantity: 100

item_id: Item002
description: Clothing
quantity: 200

item_id: Item003
description: Toys
quantity: 500

item_id: Item004
description: Statue
quantity: 50

Customers

customer_id: Cust001
customer_name: John Smith
email: john.smith@abc.com
address: 123 Elm St, City A
latitude": 41.1234,
longitude": -71.5678

customer_id: Cust002
customer_name: Alice Johnson

email: alice.johnson@xyz.com
address: "45 Maple St, City B
latitude: 35.6789,
longitude: -84.1234

13.Functionalities

The functionalities that should be included in the solution which designed to solve the issue with delivery schedule.

- **Address Geocoding**- Converting delivery address in to coordinates.
- **Multi-Stop Planning**- Planning route with multiple stops per van, Optimising the order of stops for efficiency.
- **Optimal Loading**- Determine the most effective method for loading items onto delivery vans in order to reduce unloading and reloading.
- **Historical Data Analysis and reports**- Analyze previous delivery data and reports to spot trends and enhance routing algorithms in the future.
- **Generate list of coordinates and many more**- generate a list of coordinated of the coordinates of the delivery locations to view and also create other list needed to get the information about delivery route.
- **Capacity Constraints**- Consider vehicle capacity limits to guarantee that items can be delivered in each vehicle.
- **Route Optimization**- Reduce trip distance by optimizing delivery routes.
- **Generate delivery routes**-create the full list of delivery route using the customer addresses.
- **Time Windows**- fixed time windows for the delivery.

14.Pseudo code

```
Start
   Initialize scanner for user input // Initialize scanner for user input
   Create an empty array of Customer objects called 'customers'
   Create an empty 2D array called 'cityDistanceGraph' for city distances
   Create an empty array of Order objects called 'orders'

   Output "Enter the number of cities:"
// 'numberOfCities' represents the number of cities in the input.
   Read 'numberOfCities' from user input

   For i = 0 to numberOfCities - 1
      Output "Enter the City Name for City i:"
      Read 'cityNameInput' from user input
      Convert 'cityNameInput' to uppercase and store in 'cityName'
      Output "Enter Customer ID for City i:"
      Read 'customerId' from user input
      Output "Enter Order ID for City i:"
```

```
        Read 'orderId' from user input
        Output "Enter Customer Name for City i:"
        Read 'customerName' from user input
        Output "Enter latitude for City i:"
        Read 'cityLt' from user input
        Output "Enter longitude for City i:"
        Read 'cityLn' from user input

        Create a new Customer object with 'cityName', 'customerId', 'orderId', 'customerName',
'cityLt', and 'cityLn'
        Store the Customer object in 'customers' array at index 'i'

    Create a 2D array 'cityDistanceGraph' of size 'numberOfCities' x 'numberOfCities'

    For i = 0 to numberOfCities - 1
        For j = 0 to numberOfCities - 1
            If i is equal to j
                Set 'cityDistanceGraph[i][j]' to 0
            Else
                Output "Enter the distance from City i to City j:"
                Read 'distance' from user input
                Set 'cityDistanceGraph[i][j]' to 'distance'
            End If
        Next j
    Next i

    Create an array 'orders' of Order objects

    For i = 0 to numberOfCities - 1
        Create a new Order object with 'i' as orderId and 'customers[i]' as customer
        Store the Order object in 'orders' array at index 'i'

    Initialize 'tour' as an array of integers of size 'numberOfCities'
    Initialize 'visited' as an array of booleans of size 'numberOfCities'

    Set 'tour[0]' to 0
    Set 'visited[0]' to true

    For i = 1 to numberOfCities - 1
        Set 'nearestCity' to findNearestCity('tour[i - 1]', 'cityDistanceGraph', 'visited')
        Set 'tour[i]' to 'nearestCity'
        Set 'visited[nearestCity]' to true

    Initialize 'modifiedTour' as an array of integers of size 'numberOfCities + 1'

    For i = 0 to numberOfCities - 1
        Set 'modifiedTour[i]' to 'tour[i]'

    Set 'modifiedTour[numberOfCities]' to 'tour[0]'
```

```
    Initialize an empty list called 'deliveryRoute'

    For i = 0 to 'modifiedTour' length - 1
        Set 'cityIndex' to 'modifiedTour[i]'
        Get the corresponding Customer object from 'customers' at 'cityIndex'
        Add the Customer object to 'deliveryRoute'

    Output "Delivery Route:"

    For each 'customer' in 'deliveryRoute'
        Output "City customer.getCityName() Customer ID= customer.getCustomerId(), Order
ID= customer.getOrderId(), Name= customer.getCustomerName(), lat= customer.getCityLt(),
Lon= customer.getCityLn()"

    Initialize 'totalCost' to 0

    For i = 0 to 'modifiedTour' length - 2
        Add 'cityDistanceGraph[modifiedTour[i]][modifiedTour[i + 1]]' to 'totalCost'

    Output "Total Distances: totalCost"

    Close the scanner
    Serialize the Main object and save it to a file named "main.ser"
    Output "Main object has been serialized."

End
```

15. Limitations of pseudo code

- There is no error handling or input validation in the pseudocode. In order to ensure the program's stability in real-world applications, input validation and error management are essential.
- The pseudocode lists the essential processes, however it doesn't provide in-depth explanations for some methods or functions, such as "findNearestCity" and "calculateTourCost." These functions would need more explanation and implementation information in order to be used in real life.
- The management of exceptions, which is necessary in programming to gracefully accept unforeseen failures and exceptions, is not addressed.
- Without mentioning their precise data types (e.g., int, boolean), the pseudocode employs generic data types like integers and booleans. Data types are essential for memory allocation and type safety in actual programming.
- The manner in which the code should be divided into functions and classes is not covered by the pseudocode. For readability and maintainability, code organization is crucial in real-world applications.
- The pseudocode does not explain these methods in detail, while mentioning computational stages like determining the closest city or figuring out the cost of the trip. Clear algorithmic descriptions would be necessary for real implementations.
- Concerns with efficiency or optimization are not taken into account in the pseudocode. For huge datasets in real-world applications, optimizing algorithms and data structures may be required.

- The usage of functions to encapsulate particular functionality is not highlighted in the pseudocode, nor is modularity. Modular design helps real code be reused frequently.

16. Functionalities to include in the future.

- **Dynamic Time Windows**- Allow for changing time periods based on client preferences or operational needs.
- **Constraint Handling**- Manage a variety of limitations, including vehicle maintenance schedules, parcel contraints.
- **Scalability**- Ensure that the solution can scale to handle growing volumes of delivery orders and data.
- **Vehicle Tracking**- Implement GPS monitoring for delivery vans so that their location and progress may be tracked in real time.
- **Dynamic Pricing**- Optimize revenue creation by using dynamic pricing techniques depending on parameters such as delivery distance, demand, and time of day.
- **Customer Loyalty Programs**- In order to boost repeat business and client retention, integrate customer loyalty programmes and incentives.
- **Multiple vehicle management**- having multiple van for the delivery process.
- **Vehicle Load Balancing**- Distribute packages among cars to provide a balanced workload while reducing the number of vehicles required.
- **Delivery Confirmation**- To validate successful deliveries, collect proof of delivery, such as client signatures or pictures.

17. Assumptions made for second scenario.

- Using the same door to unload the items.
- Item lastly loaded to the van will be delivered first.
- There will be only six delivery stops for at a time.
- Use the same van used for the loading parcels will be taken for the delivery.
- Visited nodes will not revisit again.
- Delivery route is started from the location of the QuickBox location.
- Delivery route will be finished after returning to starting position which is QuickBox location.
- Get the distance between each stop.
- Customer will receive the parcel in single delivery.
- No special constraints, such as urgent, fragile