

***A signed copy of this form must be submitted with every assignment.
If the statement is missing your work may not be marked.
For Level 7 assignments, please use the separate Level 7 Candidate
Statement of Own Work form instead of this one.***

Student Declaration

I confirm the following details:

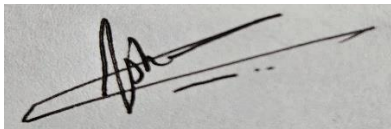
Candidate Name:	Ishan Hemant Bapardekar
Candidate ID Number:	213062
Qualification:	L4 DC
Unit:	Algorithms and mathematical concepts
Centre:	FUT004
Word Count:	2653/2500
<p>I have read and understood both NCC Education's <i>Academic Misconduct Policy</i> and the <i>Referencing and Bibliographies</i> document. To the best of my knowledge my work has been accurately referenced and all sources cited correctly.</p> <p>I confirm that I have not exceeded the stipulated word limit by more than 10%.</p> <p>I confirm that this is my own work and that I have not colluded or plagiarised any part of it.</p>	
Candidate Signature:	
Date:	28 th October 2024

Table of content

Chapter no.	title	Pg no.
1.	Task 1: Optimal Route Planning and Propositional Logic for Supply Chain Logistics	03
2.	Defining Decision Factors as Propositions	03
3.	Formulate Pseudocode	04
4.	Discuss Constraints and Alternatives	05
5.	Summary	05
6.	Task 2 - Demand Forecasting and Predicate Logic	06
7.	Define Predicates	06
8.	Applying Quantifiers:	06
9.	Predicate Modelling	06
10.	Summary	07
11.	Task 3: Visualisation Techniques and Selection of Efficient Data Structures	08
12.	Utilize Venn Diagram	08
13.	Choose Suitable Data Structures	09
14.	Implement Algorithms	11
15.	Summary	12
16.	Task 4: Algorithm Comparison for Supply Chain Logistics	13
17.	Compare different algorithms	13
18.	Analyze algorithm efficiency	14
19.	Consider practical constraints	14
20.	Summary	14

Table of figures

Fig.no	figures	Pg no.
3.1	Venn diagram of supply chain	08
3.2	Graph representation of supply chain	09
3.3	Hash table	09
3.4	Dijkstra's algorithm for supply chain	11

Task 1: Optimal Route Planning and Propositional Logic for Supply Chain Logistics

a) Defining Decision Factors as Propositions

Supply chain logistics propositions are binary conditions that influence route planning, enabling logistics teams to assess crucial factors more efficiently. Key propositions include:

Proposition P: "Vehicle is available for transportation."

Description: Checks if a vehicle is prepared for dispatch. If available, this proposition is true, allowing transportation.

Proposition Q: "Road conditions are favorable for travel."

Description: Determines if roads are open and safe. True status supports travel; otherwise, the proposition is false.

Proposition R: "Delivery deadline can be met."

Description: Assesses if the delivery can be on schedule. If true, the deadline is feasible; otherwise, alternate routes may be needed.

Proposition S: "Route is cost-efficient."

Description: Indicates the economic viability, with a true status confirming minimized expenses.

Proposition T: "Sufficient inventory is maintained in the warehouse."

Description: Ensures there is enough stock for the delivery; true means inventory is ready.

Proposition U: "Weather conditions are favorable for travel."

Description: Considers the weather; favorable conditions are true, while adverse conditions prompt review.

These propositions support swift logistical decision-making, ensuring essential conditions meet operational requirements.

b) Formulate Pseudocode

The following pseudocode checks the truth values of these propositions to determine the best route:
Algorithm OptimalRoutePlanning(P, Q, R, S, T, U)

1. Begin
2. Check if the following conditions are true:
 - P: "Vehicle is available for transportation"
 - Q: "Road conditions are favorable for travel"
 - T: "Sufficient inventory is maintained in the warehouse"
3. If all conditions (P, Q, T) are true:
 - Check if R (Delivery deadline can be met) is true
 - If R is true, check the following:
 - U: "Weather conditions are favorable for travel"
 - X: "No traffic congestion on the designated route"
 - If both U and X are true:
 - Check if S (Route is cost-efficient) is true
 - If S is true, select the **cost-efficient route**
 - If S is false, select a **standard route with deadline**
 - If U is false:
 - If U is false, display: "Unfavorable weather, review travel options"
 - If R is false:
 - Display: "Delivery deadline cannot be met on this route"
4. If any of the main conditions (P, Q, or T) are false:
 - If P is false, display: "Vehicle unavailable for transportation"
 - If Q is false, display: "Road conditions unsuitable for travel"
 - If T is false, display: "Insufficient inventory for delivery"
5. End

This pseudocode guarantees that when all propositions are true, the optimal route is chosen. If any proposition is false, it offers specific feedback, simplifying the decision-making process.

c) Discuss Constraints and Alternatives

While useful, propositional logic has limitations in logistics applications:

1. **Complexity Management:** Propositional logic tends to oversimplify complex logistics scenarios. It often overlooks how different factors, like traffic levels and road conditions, interact and affect delivery times.
2. **Inflexibility:** This approach is quite rigid and struggles to adapt to real-time changes. Sudden events, such as road closures or traffic jams, can disrupt planned routes and lead to delays.
3. **Uncertainty Handling:** Propositional logic assumes propositions are either true or false, which fails to address the uncertainties inherent in logistics. This can lead to ineffective decision-making when faced with unpredictable traffic patterns or adverse weather conditions.

Alternative Methods

1. **Heuristic Algorithms:** These algorithms utilize practical rules to find satisfactory solutions, adapting to changing conditions like anticipated traffic patterns.
2. **Constraint Satisfaction Techniques:** These methods define criteria that must be met for a solution to be valid, allowing for more complex decision-making.
3. **Machine Learning:** Analysing historical data can help predict future conditions and improve route suggestions over time.

d) Summary

In Task 1, we examined how propositional logic aids route planning in logistics. While useful, it falls short in complex situations. Employing methods like heuristic algorithms and machine learning can enhance decision-making, making supply chains more efficient and responsive to real-time challenges.

Task 2 - Demand Forecasting and Predicate Logic

a) Define Predicates

In supply chain logistics, demand forecasting is influenced by various factors over time. These factors are articulated through predicates based on specific variables.

Some key predicates for demand forecasting are:

1. Predicate $M(x)$: "Market conditions affect the demand for product x."
 - This predicates that the current state of the market, like economic trends or customer preferences, can change how much of product x is demanded.
2. Predicate $C(x)$: "Competitor actions impact the demand for product x."
 - This predicates that what competitors do, such as launching a new product or running a sale, can influence how much of product x customers want to buy. If a competitor offers a similar product at a lower price, it may decrease demand for our product.
3. Predicate $P(x)$: "Promotional strategies influence the demand for product x."
 - This predicates that marketing efforts, like discounts or advertising campaigns, can significantly boost the demand for product x. If a company runs a successful promotion, it's likely to see an increase in sales.

Example predicates:

- **Predicate $P(x)$** : "Promotional strategies influence the demand for product x."
- **Predicate $D(x)$** : "Consumer behaviour trends influence the demand for product x."
- **Predicate $S(x)$** : "Seasonal changes affect the demand for product x."

b) Applying Quantifiers:

1. Universal Quantifier (\forall):

$\forall x$ (Product x): $M(x) \wedge C(x) \rightarrow \text{forecast_increase}(x)$

Example: If all smartphones have good market conditions and strong competitor activity, then demand for each smartphone will likely increase.

2. Existential Quantifier (\exists)

$\exists x$ (Product x): $P(x) \wedge M(x) \rightarrow \text{forecast_increase}(x)$

Example: There exists a new health drink that, due to its successful marketing and positive market conditions, will see increased demand.

3. Universal Quantifier with a Negation (\neg)

$\forall x$ (Product x): $\neg D(x) \rightarrow \text{no_forecast_increase}(x)$

Example: If all snack products lack consumer trends, then none will see a demand increase.

4. Existential Quantifier with Conditions

$\exists x$ (Product x): $S(x) \wedge D(x) \rightarrow \text{forecast_increase}(x)$

Example: There's at least one winter coat that, due to seasonal changes and rising fashion trends, will experience higher demand.

c) Predicate Modelling

- Predicate logic aids in creating predictive models for demand forecasting. By leveraging historical data and external influences, these rules help generate logical predictions about future demand changes.
- Example of a Predictive Model Using Predicate Logic:
 - pseudocode

- Copy code

```
// For each product, evaluate predicates and adjust forecast accordingly for each product x in products:
```

```
if (M(x) == TRUE and C(x) == TRUE and P(x) == TRUE) then
```

```
  forecast_increase(x)
```

```
else if (M(x) == FALSE or C(x) == TRUE) then
```

```
  forecast_decrease(x)
```

```
else if (M(x) == TRUE and C(x) == FALSE and P(x) == TRUE) then
```

```
  forecast_stable(x)
```

```
end if
```

```
end for
```

```
End Algorithms
```

This algorithm dynamically adjusts forecasts based on the truth values of predicates.

For example:

When market conditions are favorable and competition suggests rising demand, the forecast indicates growth. Conversely, if the market is struggling but competition is strong, a decline is likely. In cases where the market is positive, competition is minimal, and promotions are effective, the forecast points to stable demand.

Integration of Historical Data:

In practical situations, historical demand data plays a crucial role in calibrating the truth values of predicates:

- **M(x)** is informed by past market trends.
- **C(x)** is influenced by competitive dynamics and strategies.
- **P(x)** is shaped by the effectiveness of previous promotional efforts and their impact on sales.

d) summary

This study explored how predicate logic can enhance demand forecasting within supply chains. By defining predicates such as market conditions and competitor behavior, we pinpointed key factors that influence demand. We employed quantifiers to generalize these interactions across various products, which helped in predicting demand trends. However, while predicate logic provides a solid foundation, it faces challenges when dealing with the complexities and uncertainties present in real markets. Although effective for simpler situations, more advanced techniques like machine learning may be necessary to address the unpredictable elements of consumer behavior and external factors.

These concepts greatly enhance supply chain efficiency, allowing organizations to better anticipate demand fluctuations and improve their inventory, production, and logistics strategies.

Task 3: Visualisation Techniques and Selection of Efficient Data Structures

a) Utilize Venn Diagram:

To represent relationships between different elements in a supply chain logistics network using Venn diagrams, you can depict how warehouses, distribution centers, and retail outlets are interconnected. Here's how you might use Venn diagrams for various aspects:

1. Warehouse (W), Distribution Centres (D), and Retail Outlets (R)

- W: Represents storage facilities that stock inventory.
- D: Represents distribution centres where goods are sorted and shipped out.
- R: Represents retail outlets where goods are sold to consumers.

Venn diagram:

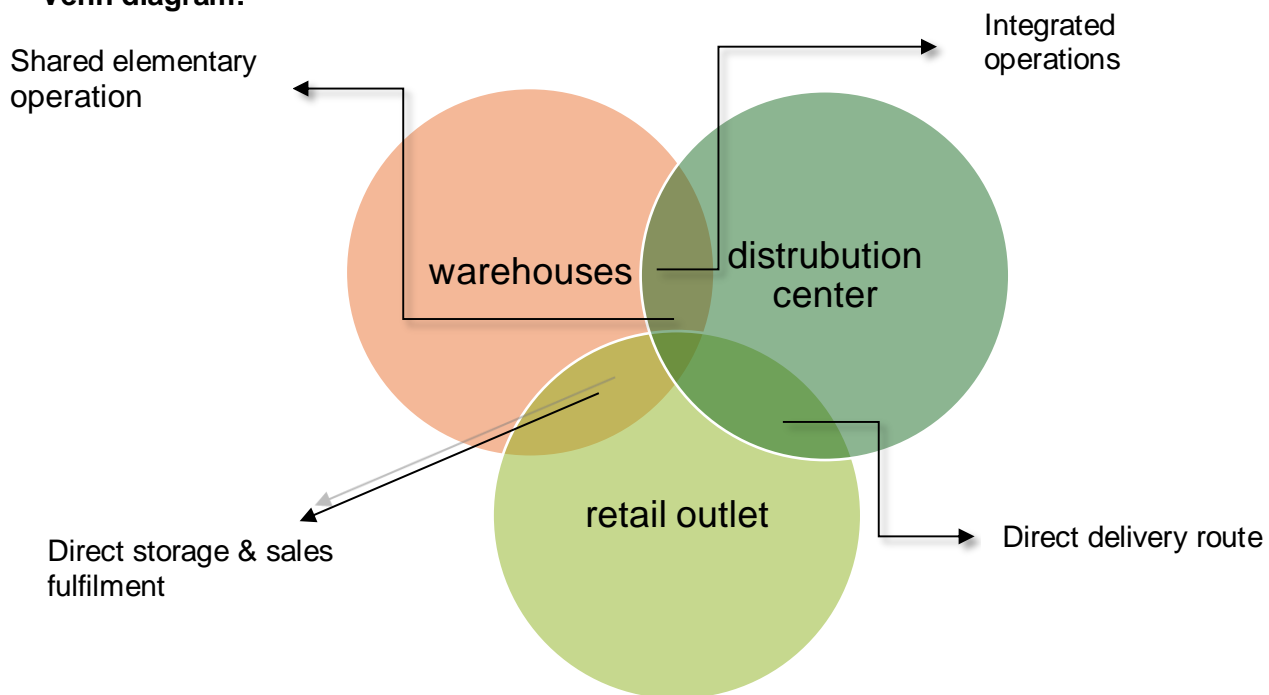


Fig- 3.1 Venn diagram of supply chain

Here the above Venn diagram represents the relationship between a warehouse, distribution centre and retail outlet

- Warehouse focuses on storing goods and managing bulk inventory.
- Distribution centre emphasizes sorting and redistributing products to various destinations.
- Retail outlet is primarily involved in direct to consumer (d2c) sales

The overlaps indicate shared functions, such as inventory management or fulfilment for retail

b) Choose Suitable Data Structures:

I. Graph:

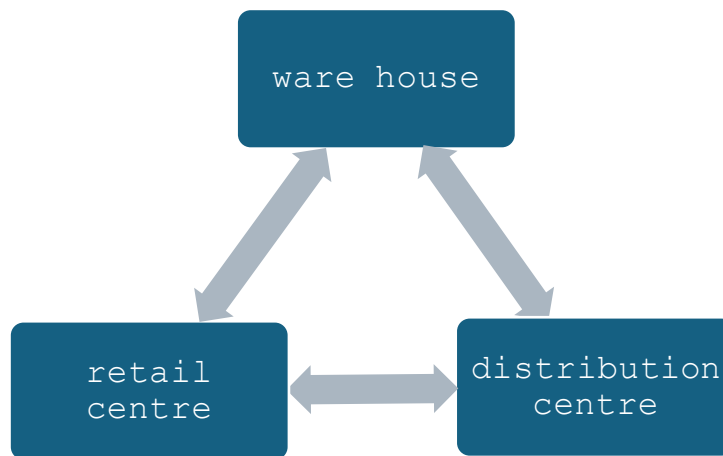


Fig-3.2 Graph representation of supply chain

This diagram represents the flow of goods between three key components in a supply chain:

1. Warehouse: This is where the goods are stored before they are distributed
2. Retail Centre: This is where the goods are eventually sold to customers.
3. Distribution Centre: This is the middle point, responsible for receiving goods from the warehouse and delivering them to the retail centres.

The arrows between them suggest movement or the transfer of goods between these locations, showing how products move through the supply chain from storage (warehouse) to sales (retail centre), with the distribution centre helping facilitate this process.

II. Hash table

Product key	Product name	Inventory level	price	Warehouse location
P001	Organic Espresso Beans	150	\$12.99	Warehouse a
P002	French Press	75	\$29.99	Warehouse b
P002	Coffee Grinder	50	\$49.99	Warehouse a
P003	Ceramic Mug	200	\$9.99	Warehouse c
P004	Coffee Subscription	100	\$19.99	Warehouse b

Fig-3.3 hash table

1. **Product Key:** A specific identifier (e.g., "P001") unique to each product. In a hash table, this key allows instant access to product data.
2. **Product Name:** The product's identifying label or description.
3. **Quantity in Stock:** Reflects the available units of the product.
4. **Price:** Shows the product's selling price per unit.
5. **Storage Location:** Identifies the location for storing each item, essential for inventory tracking.

Hash Table Basics

A **hash table** stores data as key-value pairs, allowing fast access by linking each key to a specific value. Here's a general overview of how it operates:

- **Hash Function:** Converts a key (such as "P001") into an index, guiding where to store each value in memory.
- **Handling Collisions:** When two keys produce the same index, hash tables resolve this by using methods like chaining or open addressing.
- **Optimized Efficiency:** Hash tables offer nearly constant time for insertion, deletion, and searches, making them valuable for high-frequency tasks like inventory checks.

c) Implement Algorithms:

Dijkstra's Algorithm

Shortest Route Finding with Dijkstra's Algorithm

To enhance route optimization, consider using adjustable weights to account for factors like travel time and traffic, making routes more tailored to specific needs. Path tracking allows you to record and visualize the route taken. Heuristics, like straight-line distance, can help speed up calculations on a map. Additionally, allowing users to set preferences adjusts weights for more personalized routes. For larger datasets, parallel computation can boost efficiency by running multiple route calculations at once.

Simplified Example of Dijkstra's Algorithm:

```
import heapq

def smart_dijkstra(graph, start, target):
    distances = {node: float('infinity') for node in graph}
    distances[start] = 0
    priority_queue = [(0, start)]
    previous_nodes = {node: None for node in graph}

    while priority_queue:
        current_distance, current_node = heapq.heappop(priority_queue)

        if current_node == target: # Exit early if the target is reached
            break

        if current_distance > distances[current_node]:
            continue

        for neighbor, weight in graph[current_node].items():
            distance = current_distance + weight

            if distance < distances[neighbor]: # Update if a shorter path is found
                distances[neighbor] = distance
                previous_nodes[neighbor] = current_node
                heapq.heappush(priority_queue, (distance, neighbor))

    # Reconstruct the shortest path
    path = []
    current = target
    while current is not None:
        path.append(current)
        current = previous_nodes[current]
    path.reverse()

    return distances, path # Return both distances and the route

# Example graph
graph = {
    'A': {'B': 1, 'C': 4},
    'B': {'A': 1, 'C': 2, 'D': 5},
    'C': {'A': 4, 'B': 2, 'D': 1},
    'D': {'B': 5, 'C': 1}
}

# Run the optimized algorithm
shortest_paths, route = smart_dijkstra(graph, 'A', 'D')
print("Shortest distances:", shortest_paths)
print("Route from A to D:", route)
```

Fig 3.4 Dijkstra's algorithm for supply chain

d) Summary

In Task 3, I explored several concepts that can make supply chain logistics more efficient. First, Venn diagrams help illustrate how warehouses, distribution centers, and retail outlets are interconnected, showing overlapping areas like inventory management. Graph structures then offer a clear picture of the flow of goods from storage to distribution to final retail, simplifying how we visualize and optimize these paths. Hash tables further support this by allowing quick access to product information, like stock levels and storage locations—making it easier to track inventory in real-time. Finally, Dijkstra's algorithm provides a way to find the shortest delivery routes, reducing both travel times and costs.

Together, these tools create a more streamlined supply chain. With faster data access, clear visual representations, and efficient route planning, they help ensure products are available where they're needed and meet demand effectively, all while controlling costs.

Task 4: Algorithm Comparison for Supply Chain Logistics

A. Compare different algorithms

Sorting algorithms

- **Merge Sort:** merge sort is a reliable algorithm that organizes data by dividing it into individual elements and systematically combining them in order, consistently achieving $O(n \log n)$ time while maintaining stability
 - **Quick Sort:** quick sort selects a "pivot" element to arrange items, averaging $O(n \log n)$ time but potentially degrading to $O(n^2)$. It's less stable, making it suitable for various sorting needs.
- i. **Role of quick sort in supply chain:** Quick Sort enhances supply chain efficiency by organizing inventory, prioritizing urgent orders, optimizing delivery routes, and ranking suppliers, which streamlines operations and enables swift responses to changing demand
 - ii. **Role of merge sort in supply chain:** Merge Sort strengthens supply chain management by organizing inventory, streamlining order fulfillment, uncovering trends, and ranking suppliers, ensuring smooth operations and fast adaptability to shifting demands.

Searching algorithms

- **Linear search:** Linear Search is a simple method for finding a specific value in a list or array. It checks each element sequentially until it finds a match or reaches the end. While it has a worst-case time complexity of $O(n)$, making it less efficient for large datasets, its straightforward nature makes it ideal for unsorted lists and smaller collections where ease of use is essential.
 - **Binary search:** Binary Search is an efficient method for finding a specific value in a sorted list or array. It uses two pointers to define the search range, calculates the midpoint, and compares it to the target. This process continues, narrowing down the range until the target is found. With a time complexity of $O(\log n)$, it's faster than Linear Search but requires sorted data.
- I. **Role of linear Search in supply chain:** Linear Search aids supply chain management by quickly locating items in small inventories, verifying orders, evaluating suppliers, and checking stock levels, providing straightforward, efficient support for quick tasks.
 - II. **Role of binary search in supply chain:** Binary Search enhances supply chain management by efficiently locating products in large inventories, verifying orders, comparing suppliers, and aiding data analysis, leading to quicker decisions and streamlined operations..

B. Analyze algorithm efficiency

Time complexity

1. **Sorting:** Quick Sort and Merge Sort both efficiently handle large datasets with an average time complexity of $O(n \log n)$. Merge Sort is stable and consistent, while Quick Sort is typically quicker overall.
2. **Searching:** Linear Search has a time complexity of $O(n)$, checking each element sequentially, while Binary Search achieves $O(\log n)$ by dividing the search range in half each time.

C. Consider practical constraints

When real-time data access is crucial, Binary Search is ideal for pre-sorted data, offering quick response times. For unsorted data, Linear Search might be needed, though it's less efficient. For real-time sorting, Quick Sort often delivers faster results, but for consistently large or varying datasets, Merge Sort may be a more reliable choice.

D. Summary

In supply chain logistics, choosing the right algorithms for sorting and searching is essential for maintaining efficiency and scalability. Quick Sort and Merge Sort are both effective for handling large datasets. Quick Sort generally delivers faster average performance but can be slower in the worst-case scenarios, while Merge Sort provides reliable performance, albeit with a greater memory requirement. For searching, Binary Search is perfect for quick lookups in sorted datasets due to its logarithmic time complexity, while Linear Search, though slower, is useful for unsorted or smaller datasets. When selecting algorithms, practical factors like computational resources and real-time needs come into play. Quick Sort and Binary Search are often favored for their memory efficiency and speed in larger systems. Understanding these trade-offs enables logistics managers to make informed decisions, ultimately enhancing inventory management, product sorting, and route optimization. By selecting the optimal algorithms, logistics operations can achieve faster processing, lower costs, and improved overall efficiency.

Candidate checklist

Please use the following checklist to ensure that your work is ready for submission.

- Have you read the NCC Education document *Academic Misconduct Policy* and ensured that you have acknowledged all the sources that you have used in your work? ☐
- Have you completed the *Statement and Confirmation of Own Work* form and attached it to your assignment? **You must do this.** ☐
- Have you ensured that your work has not gone over or under the recommended word count by more than 10%? ☐
- Have you ensured that your work does not contain viruses and can be run directly? ☐