Grocery Store Inventory System

Submitted By :

Name: Mehul
Course: BTech Computer Science Engineering (AI/ML)
Roll Number: 2401730266
Subject: Data Structures and Algorithms Lab Assignment 01

---

1. Introduction

This report presents the development of an Inventory System for a Grocery Store using array-based data structures in C++. The system manages stock items, tracks inventory status, supports operations such as insertion, deletion, and search, and optimizes storage for rarely restocked products using sparse representation. The implementation focuses on efficiency and complexity analysis.

---

2. Inventory Item ADT Design

The core data abstraction used is the Inventory Item (ADT) which has the following attributes:

- **ItemID (int):** Unique identifier for each item.
- **ItemName (string):** Name of the grocery item.
- **Quantity (int):** Current stock quantity of the item.
- **Price (float):** Price per unit of the item.

The following key methods are implemented:

- insertItem(data): Inserts a new item into the inventory array.
- deleteItem(ItemID): Removes an item identified by ItemID.
- searchItem(ItemID or ItemName): Searches and retrieves item details using ItemID or ItemName.

---

3. Inventory Management System

The system uses an array called ItemArray to store all inventory items. The following features are included:

- **Single and Multi-dimensional Arrays:** The item list is stored in a 1D array, while price and quantity information are managed using both row-major and column-major approaches.
- **Row-major Ordering:** Data stored row-wise, where each row represents one item with its price and quantity.
- **Column-major Ordering:** Data stored column-wise, grouping price and quantity columns separately.
- **Sparse Matrix Representation:** Rarely restocked products (with Quantity = 0) are represented sparsely to optimize storage by storing only non-zero or relevant data.

---

4. Implementation Strategy

- The system is implemented in C++ using fixed-size arrays for simplicity and to demonstrate array manipulation skills.
- Methods are designed to maintain array integrity during insertion and deletion, shifting elements as needed.
- Searching supports both ItemID and ItemName to allow versatile item lookup.
- Sparse storage outputs only the rarely restocked items, preventing wastage of space on zero-quantity items.

TIME COMPLEXITY

Function	Time Complexity	Space Complexity	Description
insertItem	O(1) average, O(n) worst	O(1)	Insert at end, potential shifts
deleteItem	O(n)	O(1)	Shifts elements to fill the gap
searchItem	O(n)	O(1)	Linear search by ID or Name

Function	Time Complexity	Space Complexity	Description
Row-major/Column-major	O(n)	O(n)	Traverse and display arrays
Sparse Storage	O(n)	O(k) ($k \leq n$)	Output only items with zero quantity

5. Complexity Analysis

---

6. System Functionality and Output

- New items can be added dynamically until the maximum capacity is reached.
- Items can be deleted by specifying their ItemID.
- Searching functionality allows retrieval by either ItemID or ItemName.
- Price and quantity data can be viewed in both row-wise and column-wise formats.
- Sparse matrix output efficiently lists only the rarely restocked items.
- The system's console output confirms each operation’s success or failure.

---

7. Sample Test Cases

Operation	Input	Expected Output
Insert Item	{ItemID=101, ItemName="Milk", Quantity=10, Price=55.0}	"Item inserted."
Insert Rare Item	{ItemID=102, ItemName="Eggs", Quantity=0, Price=70.0}	"Item inserted."
Search by Name	ItemName = "Eggs"	Found item information index printed

Operation	Input	Expected Output
Delete Item	ItemID = 101	"Item deleted."
Show Row-major Table	N/A	Table printed with all items
Show Column-major Table	N/A	Column-wise table printed
Sparse Matrix Output	N/A	Displays items where Quantity=0

---

8. Conclusion

The Inventory System successfully implements all requested features using array data structures in C++. The inclusion of sparse matrix representation optimizes storage for items that are rarely restocked, and managing price and quantity tables in row-major and column-major formats offers data organization flexibility. Complexity analysis confirms efficient function performance befitting a practical system.
