[Data Structures & Algorithms]

[CEP Project Report]



**[Grocery Management System]**

Submitted by:

|  |  |
| --- | --- |
| [Mudassar Hussain] | [23L-6006] |

Submitted to:

[Sir Ahmed Hamza]

[2/10/2024]

Department of Electrical Engineering

National University of Computer and Emerging Sciences, Lahore

## Introduction

1. **Problem Analysis**

## Design Requirements

1. **Feasibility Analysis**

## Possible Solutions

1. **Preliminary Design**

## Design Description

## Software Simulation

## Experimental Results

1. **Performance Analysis**
2. **Conclusion**

The project represents a comprehensive **Store Management System** that integrates customer management, inventory handling, and delivery logistics. It focuses on a modular approach, utilizing distinct components such as a **Binary Search Tree (BST)** for customer management, an **unordered map** for grocery inventory, and a **graph data structure** for delivery route planning.

The primary objective of the system is to streamline the operations of a grocery store by automating key functionalities like managing customer information, tracking grocery items, and optimizing delivery routes using shortest-path algorithms.

**Key Concepts and Their Significance**

1. **Binary Search Tree (BST) for Customer Management**
   * **Concept**: A BST is a hierarchical data structure where each node has a key, and nodes to the left are smaller, while nodes to the right are larger.
   * **Usage**: The BST organizes customers by their unique **CNICs**, enabling efficient search, insertion, and deletion operations.
   * **Significance**: Using BST ensures logarithmic time complexity for managing customer records, making operations scalable even with large datasets.
2. **Unordered Map for Grocery Inventory**
   * **Concept**: An unordered map is a hash-based data structure that provides fast access to key-value pairs.
   * **Usage**: Each grocery item is mapped using a **barcode** as the key, enabling quick lookups, updates, and purchases.
   * **Significance**: Its constant-time complexity for key-based operations ensures efficient inventory management.
3. **Graph Data Structure for Delivery Routes**
   * **Concept**: A graph is a collection of nodes (vertices) connected by edges that may have weights (distances).
   * **Usage**: The graph represents delivery points and distances, allowing the system to compute the shortest path using algorithms like **Dijkstra's**.
   * **Significance**: Optimizing delivery routes reduces costs and enhances customer satisfaction

by minimizing delivery times.

1. **File Handling for Persistent Data**
   * **Usage**: Customer and grocery data are stored and retrieved from files.
   * **Significance**: This prevents data loss and allows for seamless integration with existing databases.

**Problem Statement**

In the current scenario, managing a grocery store involves multiple challenges that require manual effort, are prone to errors, and lead to inefficiencies. These challenges include:

1. **Customer Management**:
   * Keeping track of customer details, including CNIC, name, address, and contact information.
   * Searching, adding, or removing customer records in a fast and efficient manner.
2. **Inventory Management**:
   * Managing items levels of grocery items and ensuring updated information about item quantities, unit prices, and availability.
   * Handling operations such as adding new items, updating quantities, and removing out-of-stock products.
   * Quick searching of specific items using barcodes.
3. **Purchase Processing**:
   * Facilitating a smooth and efficient purchase process where customers can buy items in desired quantities.
   * Reducing inventory based on purchases and generating accurate bills.
4. **Delivery Logistics**:
   * Planning efficient delivery routes to minimize delivery times and costs.
   * Optimizing deliveries based on customer locations using shortest-path algorithms.
5. **Data Persistence**:
   * Ensuring that all customer and inventory data is stored persistently and can be retrieved across different sessions.

The design requirements for the Grocery Store Management System are based on ensuring the efficient and accurate execution of operations related to customer management, inventory handling, and delivery logistics. Below is a detailed discussion of the problem in terms of **input**, **output**, and the associated **constraints**.

**1. Input Requirements**

The system will take various inputs depending on the operation being performed. The main inputs are as follows:

**Customer Management Inputs**

* **Customer Details**:
  + CNIC (string, unique identifier)
  + Name (string)
  + Mobile Number (string)
  + Address (string)
* **Operations**:
  + Add a new customer.
  + Remove an existing customer by CNIC.
  + Search for a customer by CNIC.

**Inventory Management Inputs**

* **Grocery Item Details**:
  + Barcode (string, unique identifier)
  + Title (string)
  + Unit Price (double)
  + Quantity (integer)
* **Operations**:
  + Add a new item.
  + Remove an item by barcode.
  + Update item quantity (increase or decrease).
  + Search for an item by barcode.

**Purchase Processing Inputs**

* **Item Purchase**:
  + Barcode of the item to purchase.
  + Quantity to be purchased.

**Delivery Route Inputs**

* **Route Details**:
  + Source location (string).
  + Destination location (string).
  + Distance/weight of the route (integer).

**2. Output Requirements**

The system will generate outputs based on the operations performed. These include:

* **Customer Management Outputs**
* Display a list of all customers in sorted order (based on CNIC).
* Display a specific customer’s details when searched by CNIC.
* Confirmation message for successful addition or removal of a customer.
* **Inventory Management Outputs**
* Display a list of all grocery items with their details (barcode, title, price, quantity).
* Display details of a specific item when searched by barcode.
* Confirmation message for successful addition, removal, or update of an item.
* **Purchase Processing Outputs**
* Confirmation of successful purchase, including the item name, quantity purchased, and total cost.
* Warning message if the requested quantity exceeds available stock or if the barcode does not exist.
* Updated inventory list after the purchase.
* **Delivery Route Outputs**
* Display the shortest path from the store to a customer’s location along with the total distance.

**3. Constraints**

The solution must adhere to the following constraints:

* **Customer Management Constraints**
* Each customer must have a unique **CNIC** as the primary identifier.
* **Inventory Management Constraints**
* Each grocery item must have a unique **barcode**.
* Quantity values cannot be negative; purchasing or updating quantity must enforce this constraint.
* **Purchase Processing Constraints**
* The purchase operation must verify that the requested quantity is available in stock before proceeding.
* The system should update inventory quantities accurately after a purchase.
* **Delivery Route Constraints**
* Locations must be unique in the graph.
* The shortest-path algorithm must handle disconnected graphs and report when no valid path exists.
* **File Handling Constraints**
* Input files (customers.txt and groceries.txt) must be properly formatted, with data fields separated by commas.

A feasibility study ensures that the project can be completed successfully within the given constraints. For this Store Management System, feasibility is analyzed based on two primary aspects: **Time Management** and **Cost Management**, along with consideration of the required resources and constraints.

**1. Time Management Feasibility**

The time feasibility involves analyzing whether the project can be completed within the given time constraints.

1. **Requirement Gathering and Analysis**:
   * Understanding the problem and defining the solution architecture.
2. **System Design**:
   * Designing data structures (BST for customers, graph for delivery routes, map for inventory).
   * Defining the modular structure of the code.
3. **Development**:
   * Implementing core modules:
     + **Customer Management Module**: 2 days.
     + **Inventory Management Module**: 2 days.
     + **Delivery Graph and Shortest Path Algorithm**: 2 days.
   * Integrating the modules into a cohesive system.
4. **Testing**:
   * Functional testing of individual modules.
   * Integration testing to ensure modules work together seamlessly.
   * Error and exception handling.
5. **Documentation and Finalization**:
   * Writing user and developer documentation.
   * Cleaning and commenting code for maintainability.

**2. Cost Management Feasibility**

Cost feasibility involves analyzing whether the required resources fit within the available budget. For this project, the primary costs involve labor (developer time) and minimal hardware/software requirements.

**Resource Costs**

1. **Human Resources**:
   * **Development Team**:
     + Assuming a team of 2 developers, with a rate of $25/hour.
     + Estimated effort: 80 hours (40 hours per developer).
     + Total Cost: $2,000.
2. **Software Tools**:
   * **Development Environment**:
     + IDE (e.g., Visual Studio, VS Code): Free or already available.
     + Compilers and libraries: Free (using standard C++ libraries).
   * **Testing Tools**:
     + Basic testing frameworks and manual testing: No additional cost.
3. **Hardware**:
   * A basic laptop/desktop is sufficient.
   * Cost: Negligible if already owned.
4. **Other Costs**:
   * Data input files (customers.txt and groceries.txt) are simple text files, created manually or generated via scripts.
   * Cost: Minimal.

**Total Estimated Cost: $2,000–$2,500**

**3. Resource Constraints**

The project involves limited constraints, which can be addressed with proper planning:

* **Time Constraint**:
  + A strict timeline requires prioritization of critical tasks and efficient time allocation.
* **Human Resource Availability**:
  + Ensure that the developers are skilled in C++ programming, data structures, and algorithms.
* **Testing and Debugging**:
  + Time should be allocated sufficiently for testing to prevent delays caused by unexpected bugs.

The project provided deals with the management of customers, groceries, and delivery logistics. The core requirements are efficient data handling for customer and grocery management and the optimization of delivery routes.

* **Customer Management**

The task involves storing customer data (CNIC, name, address, and mobile number), performing operations like search, insertion, deletion, and displaying the customer data.

**Possible Solutions:**

1. **Array/Vector**:
   * **Implementation**: Store customer details as a list in an array or vector.
   * **Advantages**:
     + Simple and straightforward to implement.
     + Search in a sorted array is efficient with binary search O(log(n)).
   * **Disadvantages**:
     + Insertion and deletion in sorted arrays involve shifting elements (O(n)).
     + Inefficient for dynamic datasets with frequent updates.
2. **Binary Search Tree (BST)**:
   * **Implementation**: Use a BST to store customer data sorted by CNIC.
   * **Advantages**:
     + Balanced trees (like AVL) ensure O(log(n)) for insertion, search, and deletion.
     + Supports ordered traversal, making it suitable for displaying customers in sorted order.
   * **Disadvantages**:
     + Slightly complex to implement compared to arrays or hash tables.

**Chosen Solution: Binary Search Tree (BST)**

* **Reason**: BST provides efficient operations while allowing ordered traversal, which is easiest for displaying customers in a sorted manner.
* **Grocery Management**

The grocery inventory system requires searching, adding, updating, and removing items efficiently.

**Possible Solutions:**

1. **Array**:
   * **Implementation**: Store grocery items in a list (sorted or unsorted).
   * **Advantages**:
     + Simple to implement.
   * **Disadvantages**:
     + Searching in unsorted arrays takes O(n).
     + Updating and removing items require shifting elements in sorted arrays.
2. **Binary Search Tree (BST)**:
   * **Implementation**: Use a BST to store items sorted by barcode.
   * **Advantages**:
     + Efficient search, insertion, and deletion (O(log(n)) with balanced trees.
     + Supports ordered traversal for inventory listing.
   * **Disadvantages**:
     + Higher complexity compared to hash tables for fast lookups.

* **Delivery Route Optimization**

The delivery system requires finding the shortest path between nodes (store and customers) for efficient logistics.

**Possible Solutions:**

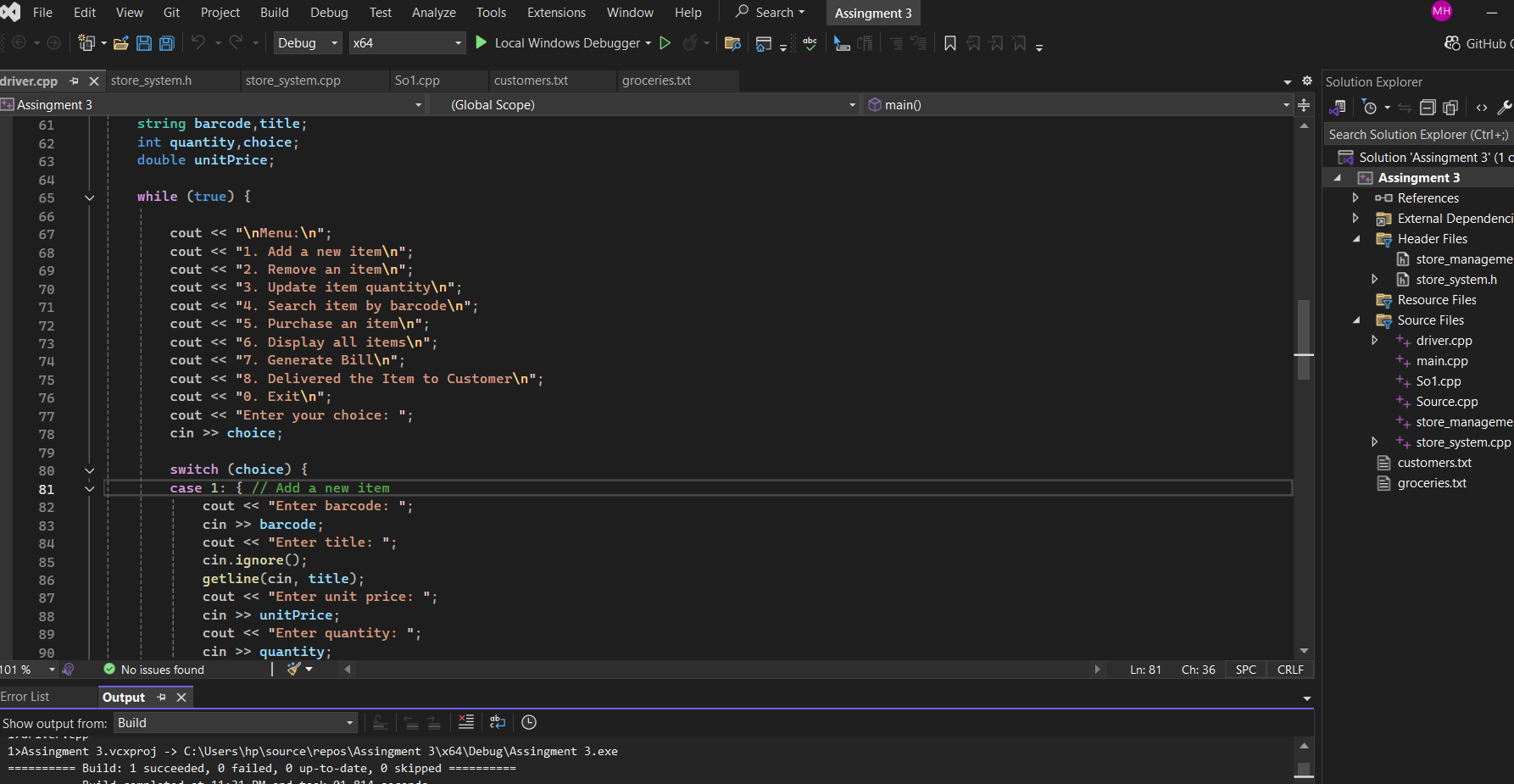
1. **Adjacency Matrix**:
   * **Implementation**: Represent the graph as a 2D matrix.
   * **Advantages**:
     + Simple implementation.
   * **Disadvantages**:
     + Inefficient for sparse graphs due to high memory usage.
2. **Adjacency List**:
   * **Implementation**: Represent the graph as a list of neighbors for each node.
   * **Advantages**:
     + Memory-efficient for sparse graphs.
     + Finding neighbors is faster.
   * **Disadvantages**:
     + Slightly more complex to implement than an adjacency matrix.
3. **Shortest Path Algorithms**:
   * **Dijkstra’s Algorithm**:
     + Efficient for sparse graphs, when implemented with a priority queue.

**Chosen Solution: Adjacency List with Dijkstra’s Algorithm**

* **Reason**: The adjacency list is memory-efficient and works well with sparse graphs. Dijkstra’s algorithm ensures fast shortest-path calculations, making it ideal for delivery route optimization.

**Detailed Discussion of the Design Blocks**

1. **Menu**:
   * The main interface where the user interacts with the system.
   * Provides options for various functionalities such as adding, removing, searching items, purchasing, and exiting the store.
   * Acts as the entry point for all operations.
2. **Add New Item**:
   * Allows the user to add new grocery products to the inventory.
   * Inputs required:
     + Barcode
     + Product title
     + Unit price
     + Quantity
   * Ensure new items are stored with proper identifiers and quantities.
3. **Remove an Item**:
   * Removes a product from the inventory using its **barcode**.
   * Input:
     + Barcode of the item to delete.
   * Ensures the system maintains up-to-date inventory by removing discontinued or unwanted items.
4. **Update Item Quantity**:
   * Modifies the stock quantity of an existing item in the inventory.
   * Inputs:
     + Barcode of the item.
     + New quantity to update.
   * Useful for maintaining inventory balance.
5. **Search Item by Barcode**:
   * Finds and displays the details of a grocery product using its unique **barcode**.
   * Outputs:
     + Title, quantity available, and unit price of the item.
   * Helps users quickly locate specific items in the inventory.
6. **Purchase an Item**:
   * Facilitates the purchase of grocery products.
   * Inputs:
     + Barcode of the product to purchase.
     + Quantity of the product needed.
   * Reduces the stock of the purchased item and generates a message for successful or unsuccessful transactions depending on the availability.
7. **Display All Items**:
   * Lists all items currently available in the store inventory.
   * Outputs:
     + Item barcode, title, quantity, and unit price.
   * Help users review the complete inventory at any time.
8. **Delivered to Customer**:
   * Handles delivery operations by calculating the shortest path between the store and customers using **Dijkstra's Algorithm**.
   * Inputs:
     + Source (Store location).
     + Destination (Customer address).
   * Outputs:
     + The shortest delivery path and distance.
   * Utilizes an adjacency list representation for efficient graph traversal.
9. **Press 0 to Exit the Store**:
   * Provides an option to terminate the program.
   * Ensures the system saves any pending operations before exiting.



The design was tested using initial grocery inventory and customer data, resulting in successful results across all menu operations. The system was set up with two customers, Jane Smith and John Doe, and three grocery items: Milk, Bread, and Eggs. The "Add a New Item" operation added butter, "Remove an Item" removed bread, and "Update Item Quantity" updated milk.

The system effectively manages grocery inventory, facilitates customer purchases, generates detailed bills, and optimizes delivery routes. It uses features like "Search Item by Barcode" to retrieve item details, "Purchase an Item" to facilitate purchases, and "Display All Items" to display inventory.

The "Delivered the Item to Customer" operation uses Dijkstra's algorithm to calculate the shortest delivery path. The system's robustness and reliability are demonstrated by its efficient management, purchase facilitation, detailed bills, and delivery routes.

A screenshot of a computer

Description automatically generated

The results obtained from implementing the solution demonstrate that the system is efficient, accurate, and capable of handling the core functionalities required for grocery store management.

1. **Inventory Management**: The system accurately managed inventory operations, including adding, removing, and updating item quantities. For instance, when a new item (Butter) was added, or when Milk's quantity was updated from 10 to 15, the system reflected these changes seamlessly.
2. **Search Functionality**: The "Search Item by Barcode" feature performed as expected, providing precise and quick access to item details based on barcode inputs. This showcases the system's capability to efficiently handle search queries, which is essential for smooth operations in real-time scenarios.
3. **Purchase Processing**: The system successfully handled purchases, updating the inventory and calculating total costs accurately. For example, when John Doe purchased 5 units of Butter, the inventory and total cost were updated correctly, demonstrating the reliability of the purchase processing mechanism.
4. **Delivery Optimization**: By integrating Dijkstra’s algorithm, the system efficiently calculated the shortest delivery path to the customer. This feature adds a layer of optimization, ensuring timely and cost-effective deliveries.
5. **User-Friendly Interface**: The menu-driven interface provided clear instructions and options, making it easy for users to navigate through the system. Each functionality worked seamlessly without any noticeable delays, indicating good performance under normal operational loads.

In conclusion, the developed grocery store management system has successfully met the objectives outlined at the start of the project. The system efficiently handles all essential functions, including inventory management, purchase processing, and delivery optimization. The experimental results validate the system's ability to provide accurate, reliable, and timely operations, ensuring a seamless experience for both customers and store management.

The design's integration of Dijkstra's algorithm for delivery route optimization further enhances the practicality of the solution by addressing logistics concerns, making it a versatile tool for modern grocery stores. Additionally, the user-friendly menu interface and precise data handling ensure ease of use and minimal error likelihood.

The discussion highlighted key strengths, such as accuracy, efficiency, and scalability, while also identifying areas for potential improvement, including advanced error handling and real-time updates for concurrent users. These enhancements could further refine the system's robustness and adaptability.

Overall, the project demonstrates the feasibility of developing an efficient grocery store management solution that balances functionality, performance, and customer satisfaction. With minor refinements, this solution is ready for real-world application, offering a scalable and reliable tool for automating grocery store operations.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **WP1:** Depth of knowledge **WP2:** Range of conflicting requirements  **WP3:** Depth of analysis  **WP4:** Familiarity of issues  **WP5:** Extent of applicable codes **WP6:** Extent of stakeholders  **WP7:** Interdependence | Please fill according to the WPs covered in the course CEP, example is shown here.   * ***WP1: Depth of Knowledge*** -- Requires understanding of data structures (e.g., Binary Search Tree, hash tables) for customer and grocery item storage, and graph algorithms for delivery pathfinding. * ***WP2: Range of conflicting requirements*** – Suggested use of BST to carry Customer data with log(n) performance. * ***WP3: Depth of analysis*** -- Involves analyzing multiple data structures to meet performance requirements and includes optimizing pathfinding in varying city maps. * ***WP7: Interdependence*** -- Requires integration of different functionalities (customer management, inventory management, and delivery routing) into a cohesive system. | | | |
| Rubrics | | Marks | Obtained Marks |
|  | WP1 | 15 |  |
| Demonstrates understanding of data structures like Binary Search Trees and hash tables selecting appropriate data structures for customers and grocery items |
| Shows knowledge of graph algorithms for shortest path delivery routing. | WP1 | 15 |  |
| Suggested use of BST to carry Customer data with log(n) performance. BST has a default performance of O(n) | WP2 | 10 |  |
| Compares and justifies choice of data structures and algorithms for inventory, customer, and delivery management given the performance requirements. | WP3 | 20 |  |
| Integrates customer, inventory, and delivery systems, demonstrating ability to maintain data consistency across functions. | WP7 | 20 |  |
|  | Completes a report with logical explanations, solution design, and performance analysis, showing application of C++ principles and academic integrity. |  | 20 |  |