Design Document

05 - ONLINE MARKETPLACE INVENTORY & ORDER MATCHING ENGINE

[Bilal Azfar - Huzaifa Soomar]

1. Introduction

This document describes the design and implementation of an Online Marketplace System developed as part of a Data Structures and Algorithms (DSA) final project. The primary goal of the system is to demonstrate the practical application of core data structures such as hash maps, heaps (priority queues), trees, and graphs in a realistic problem domain.

The system models an order-driven marketplace where buyers and sellers place orders, a matching engine pairs compatible orders based on priority rules, and a recommendation engine suggests products based on historical co-purchase patterns.

2. System Objectives

The key objectives of the system are:

- To manage a product catalog efficiently using hashing
- To represent product categories hierarchically using trees
- To implement an order book using priority queues
- To match buy and sell orders using price–time priority
- To recommend related products using graph-based analysis
- To provide an interactive command-line interface for demonstration

The system is designed for educational purposes and focuses on correctness, clarity, and alignment with DSA concepts rather than full commercial realism.

3. High-Level Architecture

The system is divided into the following major modules:

- Catalog Module

- Order Book Module
- Matching Engine
- Recommendation Engine
- Reporting Module
- Data Loading Module
- User Interface (CLI)

Each module is responsible for a distinct concern, following separation of concerns and modular design principles.

4. Catalog and Category Management

The catalog module manages products and their organization into categories.

4.1 Category Representation

- Categories are implemented using a general tree structure.
- Each category node can have multiple child categories.
- A single ROOT node serves as the top-level parent for all categories.
- Recursive traversal is used for searching and printing the category hierarchy.

This structure allows flexible nesting of categories such as Electronics → Mobile or Clothing → Jackets.

4.2 Product Storage

- Products are stored using a custom HashMap with product ID as the key.
- This enables average-case O(1) lookup, insertion, and deletion.
- A secondary list of products is maintained for iteration and reporting purposes.

Product attributes include ID, name, category path, price, and stock metadata.

5. Order Book Design

The order book manages buy and sell orders using priority queues.

5.1 Order Representation

Each order contains:

- Order ID
- Product ID
- Order type (BUY or SELL)
- Quantity
- Price
- Timestamp

The timestamp is used for tie-breaking when prices are equal.

5.2 Priority Queues

- BUY orders are stored in a max-heap
- SELL orders are stored in a min-heap
- Both heaps are implemented using a custom generic Heap data structure

Priority is determined by:

1. Price (higher buy price or lower sell price has higher priority)
2. Time (earlier orders have higher priority when prices are equal)

This enforces standard price–time priority used in order-driven markets.

6. Matching Engine

The matching engine is responsible for pairing compatible buy and sell orders.

6.1 Matching Strategy

- Matching is performed per product ID.
- Only orders for the same product are considered.
- Orders are matched when buy price is greater than or equal to sell price.
- Partial order fulfillment is supported.

6.2 Partial Fills

- If one order has a larger quantity, the matched portion is executed.
- The remaining quantity is reinserted into the order book.
- Fully filled orders are removed from both the order book and order index.

6.3 Order Indexing

- A HashMap is used to index orders by order ID.

- This allows direct lookup of orders, which is not possible efficiently using heaps alone.
- The index is updated dynamically as orders are partially or fully matched.

7. Recommendation Engine

The recommendation engine suggests products that are frequently bought together.

7.1 Graph Representation

- A weighted undirected graph is used.
- Each node represents a product.
- An edge between two products indicates they were bought together.
- Edge weight represents the frequency of co-purchase.

The graph is implemented using hash maps for adjacency lists and weight storage.

7.2 Recommendation Generation

- For a given product, its neighboring products are retrieved.
- Neighbors are sorted by descending edge weight.
- The top K products are returned as recommendations.

Recommendations are based on historical data loaded at startup and do not update dynamically during runtime.

8. Reporting and User Interface

8.1 Reports

The reporting module provides:

- Inventory listing (product ID, name, category, price)
- Best BUY and SELL orders (top of order book)
- Match execution logs during order matching

Stock is intentionally not displayed to avoid implying enforcement that the system does not implement.

8.2 Command-Line Interface

The system provides an interactive CLI allowing users to:

- View category hierarchy
- Browse inventory
- Lookup products by ID
- Place buy and sell orders
- View best orders
- Lookup orders by ID
- Run order matching for a product
- View product recommendations

9. Data Loading

- Initial data is loaded from text files at program startup.
- Data includes categories, products, orders, and co-purchase pairs.
- All data is maintained in memory during execution.
- No persistence is performed after runtime.

This approach cleanly separates configuration data from program logic.

10. Design Decisions and Scope

Several deliberate design decisions were made:

- The system is order-driven rather than inventory-driven.
- Product stock is treated as informational metadata only.
- Inventory stock is not enforced during order matching.
- Recommendations are based on historical data and are not updated during matching.
- Priority queues are used instead of FIFO queues, as required for correct order matching behavior.
- Core data structures (HashMap and Heap) are implemented from scratch to meet DSA objectives.

These decisions ensure the system remains focused on demonstrating data structures and algorithms rather than full commercial complexity.

11. Complexity Analysis

- Product lookup: O(1) average
- Order insertion: O(log n)
- Best order retrieval: O(1)
- Order removal: O(log n)
- Category search: O(n)
- Recommendation generation: O(V log V), where V is the number of related products

12. Conclusion

The Online Marketplace System successfully demonstrates the application of fundamental data structures and algorithms in a cohesive and realistic problem setting. Through modular design, custom implementations, and clear separation of concerns, the project meets all stated requirements while remaining aligned with academic learning objectives.

The system provides a solid foundation that can be extended further, but in its current form serves as a complete and correct DSA-focused marketplace simulation.