2025 Summer - Algorithms and Data Structures (MSCS-532-B01) - Second Bi-term

Dynamic Inventory Management System: Proof of Concept Implementation

Amit Yadav

ayadav36753@ucumberlands.edu

Dr. Vanessa Cooper

University of the Cumberlands

1. Partial Implementation Overview

For Phase 2, a proof of concept implementation of the Dynamic Inventory Management System was developed using core data structures designed in Phase 1. The system revolves around Python's built-in `dict` for fast product lookup by ID and `defaultdict(list)` from the `collections` module to manage category-based organization.

The primary functions implemented include:

- add_product: Inserts a new product into the inventory.
- update_product: Updates attributes such as price, quantity, or name.
- delete_product: Removes a product from the inventory.
- get_product: Retrieves product details by ID.
- list_by_category: Lists all products in a given category.

These implementations ensure constant time complexity (O(1)) for insertions and updates and linear time (O(n)) for category filtering, which suits small- to mid-scale inventory systems.

2. Demonstration and Testing

A simple CLI-based script demonstrates the working of all critical operations. The following test cases validate the implementation:

Test Case 1: Add Products

add_product(inventory, "P001", "Laptop", 799.99, 10, "Electronics")

add_product(inventory, "P002", "Coffee Mug", 9.99, 100, "Kitchenware")

Test Case 2: Update Product

update_product(inventory, "P001", price=749.99, quantity=8)

Test Case 3: Delete Product

delete_product(inventory, "P002")

Test Case 4: Get Product

get_product(inventory, "P001")

Test Case 5: List by Category

list_by_category(inventory, "Electronics")

The test results confirmed accurate insertion, retrieval, updating, and deletion operations. Invalid operations such as retrieving a non-existent product or updating with invalid values were handled gracefully using error messages.

3. Implementation Challenges and Solutions

Challenge 1: Error Handling and Validation

Initially, inserting products with duplicate IDs or updating with negative quantities was not prevented. This was addressed by adding checks:

```
if product_id in inventory:
        raise ValueError("Duplicate Product ID")
```

Challenge 2: Category Management

Products removed from inventory still left residual references in the defaultdict. This was fixed by synchronizing both dictionaries during deletion.

Challenge 3: Data Consistency

A product might be updated with a different category, breaking consistency. To fix this, updates to categories now include proper category reassignment.

4. Next Steps

To transform the proof of concept into a complete application, the following steps are recommended:

1. Persistent Storage Integration: Connect the system to an SQLite or PostgreSQL backend.
2. Concurrency Handling: Add multi-threading locks or use a database with transactional support.
3. User Interface: Develop a web or GUI-based frontend using Flask or Tkinter.
4. Advanced Search and Sort: Implement price-based and quantity-based sorting using heapq or sorting algorithms.
5. Unit Testing Framework: Integrate unittest or pytest for structured testing.

5. Code Snippets and Documentation

Add Product

```
def add_product(inventory, product_id, name, price, quantity, category):
        if product_id in inventory:
```

```
        print("Error: Product ID already exists.")
        else:
        inventory[product_id] = {
        "name": name,
        "price": price,
        "quantity": quantity,
        "category": category
        }
```

Update Product

```
def update_product(inventory, product_id, **kwargs):
        if product_id in inventory:
        for key, value in kwargs.items():
        if key in inventory[product_id]:
            inventory[product_id][key] = value
        else:
        print("Error: Product not found.")
```

Delete Product

```
def delete_product(inventory, product_id):
        if product_id in inventory:
        del inventory[product_id]
        else:
```

```
        print("Error: Product not found.")
```

List by Category

```
def list_by_category(inventory, category):
        return [p for p in inventory.values() if p["category"] == category]
```

GitHub Repository:

https://github.com/amityadav137/Project-Phase-2-Deliverable-2-Proof-of-Concept-Implementation

6. References

Skiena, S. S. (2020). The Algorithm Design Manual (3rd ed.). Springer.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.

Van Rossum, G., & Drake, F. L. (2009). The Python Language Reference Manual. Network Theory Ltd.