Inventory Management Using Optimized Data Structures in Python – Phase 2

Unique Karanjit

University of the Cumberlands

MSCS 532-A01: Algorithms and Data Structures

Dr. Satish Penmasta

February 2, 2025

Introduction

This project involves the partial implementation of two basic data structures, namely:

- *HashTable*: Implemented to efficiently store and fetch product information by using unique product IDs.
- *AVL Tree*: A Self-balancing binary search tree for sorted product information based on pricing.

These data structures are designed to be part of an inventory management system where quick lookups, insertions, and deletions are key in managing product details.

HashTable Implementation

A hash table is implemented here using separate chaining to handle collisions. That is, in case many keys hash to the same index, they are kept in a linked list at this index. The core methods implemented are as follows.

i. insert(key, product) → Adds a product to the hash table. If the key already exists, it updates the value.
ii. get(key) → Retrieves a product based on its ID.
iii. delete(key) → Removes a product from the hash table using its ID.
iv. items() → Returns all key-value pairs in the table.

AVL Tree Implementation

An AVL Tree is a self-balancing binary search tree where the height difference (balance factor) between left and right subtrees is maintained within -1 to 1. The core methods implemented are as follows.

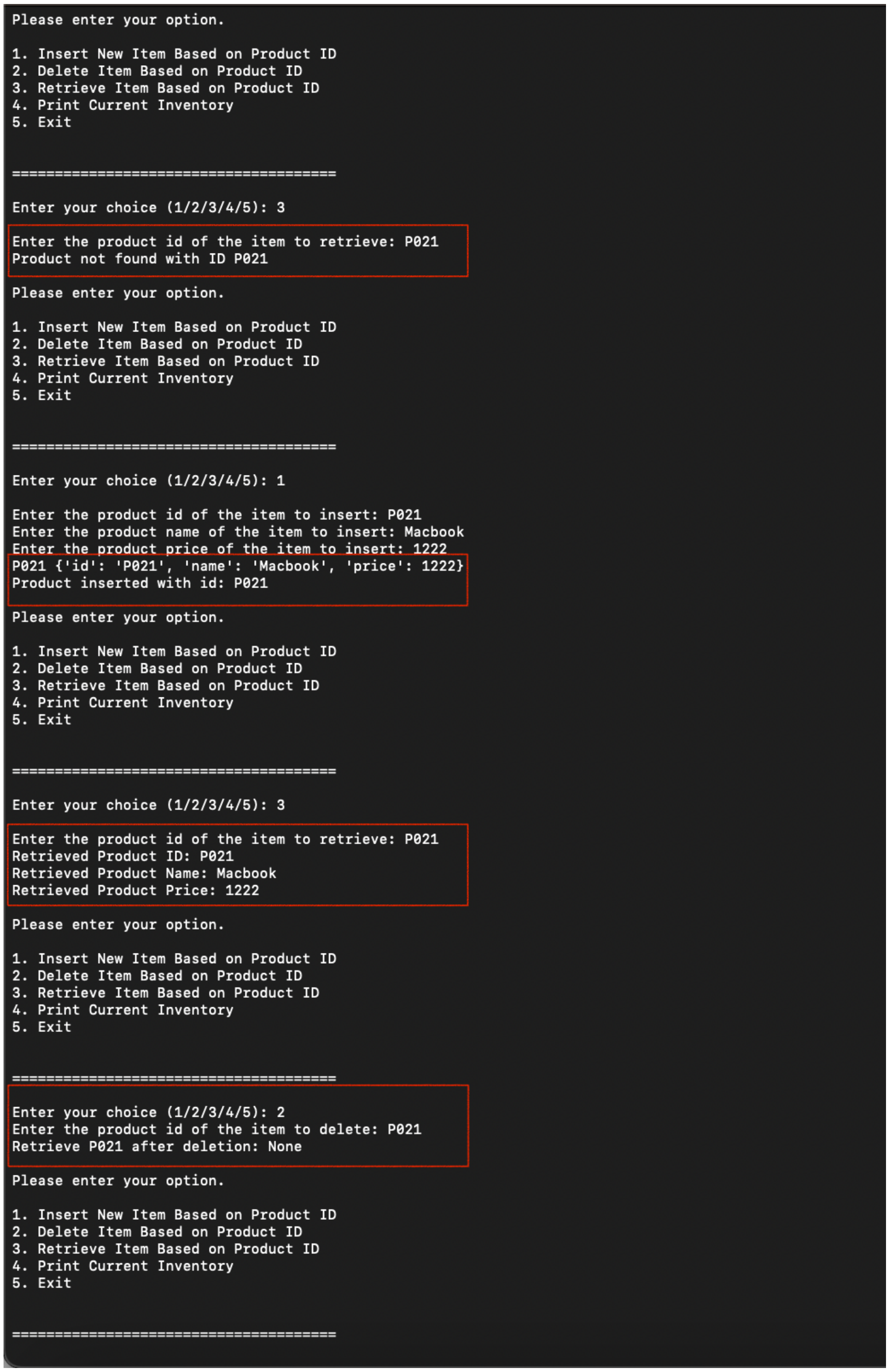
i. insert (root, price, product) → This method is responsible to insert a product into the AVL tree, keeping it balanced.
ii. balance(root) → This method ensures that the tree remains balanced after insertion.
iii. height(root) → This method computes the height of a node, which is crucial for balance checking.

Demonstration and Testing

The code is designed to manage the inventory of products. It uses an **id** field as the primary key, which is unique for each product. When any product is inserted into the system, three pieces of information are necessary: id, name, and price. The id makes each product unique, thus getting rid of duplication and resulting in easy identification of any product.

The id field is used to look up the product details in the hash table for product retrieval. This helps the system to retrieve information on a product even in cases of many entries. The AVL tree will store products sorted mainly by price for efficient search and ordering based on attributes.

For deletion, the system uses the id to locate and remove the product from both the hash table and the AVL tree. In this way, it is completely deleted from the system, and coherence between all the different data structures that are in use is guaranteed. In general, the usage of id as the main key guarantees the uniqueness of each product and, on the other hand, makes operations of insert, retrieve, and delete in inventory efficient and with no loss.



```
Please enter your option.

1. Insert New Item Based on Product ID
2. Delete Item Based on Product ID
3. Retrieve Item Based on Product ID
4. Print Current Inventory
5. Exit


======================================

Enter your choice (1/2/3/4/5): 3

Enter the product id of the item to retrieve: P021
Product not found with ID P021

Please enter your option.

1. Insert New Item Based on Product ID
2. Delete Item Based on Product ID
3. Retrieve Item Based on Product ID
4. Print Current Inventory
5. Exit


======================================

Enter your choice (1/2/3/4/5): 1

Enter the product id of the item to insert: P021
Enter the product name of the item to insert: Macbook
Enter the product price of the item to insert: 1222
P021 {'id': 'P021', 'name': 'Macbook', 'price': 1222}
Product inserted with id: P021

Please enter your option.

1. Insert New Item Based on Product ID
2. Delete Item Based on Product ID
3. Retrieve Item Based on Product ID
4. Print Current Inventory
5. Exit


======================================

Enter your choice (1/2/3/4/5): 3

Enter the product id of the item to retrieve: P021
Retrieved Product ID: P021
Retrieved Product Name: Macbook
Retrieved Product Price: 1222

Please enter your option.

1. Insert New Item Based on Product ID
2. Delete Item Based on Product ID
3. Retrieve Item Based on Product ID
4. Print Current Inventory
5. Exit


======================================

Enter your choice (1/2/3/4/5): 2
Enter the product id of the item to delete: P021
Retrieve P021 after deletion: None

Please enter your option.

1. Insert New Item Based on Product ID
2. Delete Item Based on Product ID
3. Retrieve Item Based on Product ID
4. Print Current Inventory
5. Exit


======================================
```

Implementation Challenges and Solutions Challenge

Challenge 1: Handling Collisions in HashTable

Problem: More than one product hashed to the same index; it caused value overwriting.

Solution: Used separate chaining where values at each index are stored in a list.

Challenge 2: Self-Balancing of AVL Tree

Problem: The tree becomes unbalanced when elements are inserted sequentially.

Solution: Added height tracking and created an empty method _balance() that can be used later to introduce rotational logic.

Challenge 3: Efficient Deletion in HashTable

Problem: Removal of an item at the hash index had to iterate over a list.

Solution: Utilized Python's del to remove items efficiently.

Next Steps

The following functionalities need to be added to the existing code.

- *Automate Product ID Generation:* Currently, the user must manually enter a product ID when adding a product to the hash table.Instead, the system should automatically generate unique product IDs dynamically based on predefined rules (e.g., a sequence number, timestamp, or UUID). This will reduce user errors and ensure consistency across the dataset.

- *Sorting and Price-Based Queries*: Right now, the AVL tree only supports insertion, and there is no implemented sorting mechanism. To enhance functionality, the following methods should be added:
    - Find Cheapest Product: Retrieve the leftmost node in the AVL tree (smallest price).
    - Find Most Expensive Product: Retrieve the rightmost node in the AVL tree (highest price).
    - Find Products Within a Price Range: Implement an in-order traversal to filter products within a specified price range.
    - Sort Products in Ascending/Descending Order: Use in-order traversal (ascending) and reverse in-order traversal (descending) to list products based on price.

  These features will improve product retrieval efficiency and allow for better decision-making in an inventory system.

- *Integrate with an Inventory System*: Combine both the hash table (for quick lookups) and AVL tree (for ordered retrieval) to create a complete inventory management system.

Therefore, using id as the primary key, the program provides convenient and precise processing of the product information within the inventory management system. It uses the id field to guarantee efficiency and consistency in the data-retrieving process. Also, concerning various boundary conditions like id duplication, no product existence, and illegal input, it provides integrity in the data without leading to errors while in operation.

The source code for this project can be found at - https://github.com/uniquekaranjit/InventoryManagementSystem

References

Avl Trees. CS Kenyon. (n.d.). https://cs.kenyon.edu/index.php/scmp-218-00-data-structures/avl-trees/

Enoch, Olanite & Joseph, Oloyede & Ok, Emmanuel & Williams, Barnty & Aria, Javiera & Jose, Dylan & Diego, Catalina. (2024). Data Structures and Algorithms: Trees, graphs, and hash tables, Sorting and searching algorithms.

Perez, H. D., Hubbs, C. D., Li, C., & Grossmann, I. E. (2021). Algorithmic Approaches to Inventory Management Optimization. *Processes*, *9*(1), 102. https://doi.org/10.3390/pr9010102

Walmart Global Tech. (2023). *Walmart's AI-powered inventory system brightens the holidays.* https://tech.walmart.com/content/walmart-global-tech/en_us/blog/post/walmarts-ai-powered-inventory-system-brightens-the-holidays.html