

19CSE212

Data Structures and Algorithms

CASE STUDY -HYBRID DATA STRUCTURES

TITLE: E-COMMERCE USING HYBRID DATA STRUCTURES

TEAM MEMBERS:

|  |  |  |
| --- | --- | --- |
| S.NO | NAME | ROLL NO |
| 1 | DEEPTHIKA R | CB.EN.U4CSE21217 |
| 2 | DHARSAN S | CB.EN.U4CSE21219 |

|  |
| --- |
| TABLE OF CONTENT |
| 1. INTRODUCTION |
| 1. OBJECTIVE |
| 1. IMPLEMENTATION |
| 1. GITHUB REPOSITORY LINK |
| 1. PRACTICAL APPLICATION |
| 1. PERFORMANCE ANALYSIS |
| 1. EXPERIMENTAL EVALUATION |
| 1. DISCUSSION |
| 1. CONCLUSION |
| 1. REFERENCE |

**INTRODUCTION:**

**What are hybrid Data structures?**

Hybrid data structures are those with 2 or more different types of data structures.

They are designed to enhance the strengths of each individual data structure while mitigating their weaknesses.

For example, a common hybrid data structure is the hash table with linked lists. Hash tables provide fast average case lookup time, but in the worst case, the lookup time can become linear. Linked lists, on the other hand, have a slower average case lookup time, but their worst-case lookup time is constant. By combining the two, we get a data structure that has a fast average case lookup time and worst-case performance.

**Significance of hybrid data structures in solving complex problems efficiently:**

Hybrid data structures are often used in situations where a single data structure cannot meet all the requirements of a given problem. By combining different data structures, we can create a specialized data structure that is specific to the needs of the problem at hand.

Hybrid data structures can be more complex than their individual components but are better in performance than a single data structure. They can be used in a variety of applications, including databases, file systems, and search engines.

**Improved performance:** Hybrid data structures can often improve the performance of operations such as insertion, deletion, and searching. This is because they can take advantage of the strengths of multiple data structures.

**Enhanced functionality**: Hybrid data structures can often provide enhanced functionality that is not available with traditional data structures. For example, a hash tree can provide efficient searching of large amounts of data, while a skip list can provide efficient insertion and deletion of elements in a sorted list.

**Increased flexibility**: Hybrid data structures can be more flexible than traditional data structures. This is because they can be adapted to a wider range of problems.

Overall, hybrid data structures can provide a powerful tool for solving complex data management problems by combining the strengths of different data structures to achieve better performance or address specific limitations.

**Here are some of the challenges of using hybrid data structures:**

**Complexity:** Hybrid data structures can be more complex to implement and understand than traditional data structures. This is because they combine the features of multiple data structures.

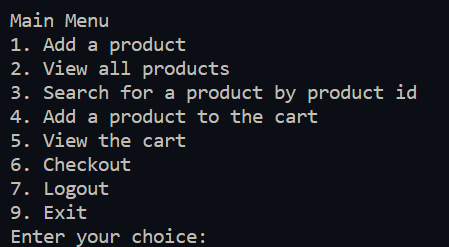
**Inefficiency:** Hybrid data structures can sometimes be less efficient than traditional data structures. This is because they may not be able to take full advantage of the strengths of all of the data structures that they combine.

**Lack of support:** Hybrid data structures may not be as widely supported as traditional data structures. This can make it difficult to find libraries and tools that support them.

**OBJECTIVE:**

**DESIGNING:**

We have designed an e-commerce platform where the users should register and then log in to use. They can enter their choices from a menu-driven list, there are features like adding new products, adding the product to the cart, searching for a particular product with its product id, checking out the product, logout from the user account, etc.



**HYBRID DATA STRUCTURES USED:**

* Hash Table
* Binary Search Tree
* Linked List

**PERFORMANCE ANALYSIS:**

**TIME COMPLEXITY:**

* adding a product to the hash table is O(1).
* adding a product to the binary search tree is O(log n), worst case O(n), where n is the number of nodes in the tree. Here n is products.
* searching for a product by pid in the hash table is O(1) in the average case and O(n) in the worst case, where n is the number of products with the same pid. Here n is products.
* adding a product to the cart is O(n), where n is the number of products with the given pid.
* viewing the cart is O(n). Here n is products.
* checking out is O(1).
* logging in or registering is O(n), where n is the number of users.

**SPACE COMPLEXITY:**

* space complexity for the hash table is O(n), where n is the number of unique product ids.
* space complexity for the binary search tree is O(n), where n is the number of products.
* space complexity for the cart is O(n), where n is the number of products in the cart.
* space complexity for the list of users is O(n), where n is the number of registered users.
* Therefore, the overall space complexity of the program depends on the number of products, users, and graph nodes and edges, and can range from O(n) to O(n^2) in the worst case.

Used as hybrid data structure compared while used separate,

The hash table and binary search tree are used to store products by their ID. The hash table provides O(1) average case insertion, retrieval, and deletion time complexity, while the binary search tree provides O(log n) for the same operations. The hybrid data structure combines the strengths of both data structures to provide optimal performance for these operations.

**OVERVIEW OF THE HYBRID DATA STRUCTURE:**

* Combination of hash table and binary search tree to efficiently store and search for Product objects. The hash table provides constant time lookup of products based on their product id, while the binary search tree allows for efficient traversal and sorting of products based on their product id.

**CHOSEN HYBRID DATA STRUCTURE**:

* Hash Table: Used to store the Product objects with their unique product id as the key. It also uses linked list to store the product details. The product id acts as a key and its value or the product details are linked using linked list.
* Binary Search Tree: Used to store the Product objects based on their product id. These products will be stored in such a way that it will be easy to retrieve any product, given that even in worst case.
* LinkedList: Used to implement a shopping cart as a linked list of Product objects. The products are linked to one and other. It is also used by the hash table to store its value or the product details, which are linked together.

**ADVANTAGES AND MOTIVATION:**

**Improved performance**: By combining the strengths of multiple data structures, hybrid data structures can often provide better performance than individual structures.

**Reduced memory usage:** Hybrid data structures can often reduce memory usage compared to using multiple data structures.

**Flexibility:** Hybrid data structures can be customized to specific use cases and can be optimized for specific operations. In the above code, the data structures used can be optimized for fast lookups and sorting of products

**Scalability:** Hybrid data structures can often scale better than individual structures.

**Simplified code:** Hybrid data structures can often simplify code by eliminating the need for multiple data structures and reducing the complexity of the code.

**PRACTICAL APPLICATIONS:**

The practical application is the development of an online shopping system that is fast, efficient, and user-friendly. This concept can be integrated into the backend of the website or application to provide users with a seamless shopping experience.

the use of a hybrid data structure in the above code provides several advantages, including improved performance, reduced memory usage, flexibility, scalability, and simplified code. By combining the strengths of multiple data structures, the hybrid structure can efficiently handle the specific requirements of an online shopping cart system.

In addition to the basic functionality provided in the code, the implementation could be extended to include additional features such as displaying related products or recommending products based on the user's purchase history. The code could also be optimized for performance by using other hybrid data structures or by implementing caching mechanisms to reduce database queries and improve response times.

**Databases:** Hybrid data structures can be used to optimize query performance in databases. For example, a combination of a hash table and a B-tree can be used to efficiently search for data in a database.

**Networking:** Hybrid data structures can be used in network routing algorithms. For example, a combination of a graph and a heap can be used to find the shortest path in a network.

**Computer graphics:** Hybrid data structures can be used in computer graphics to efficiently represent 3D objects. For example, a combination of a quadtree and an octree can be used to represent objects at different levels of detail.

**Machine learning:** Hybrid data structures can be used in machine learning algorithms to efficiently store and retrieve large datasets. For example, a combination of a hash table and a bloom filter can be used to efficiently store and query large sets of data.

**Operating systems**: Hybrid data structures can be used in operating systems to optimize file system performance. For example, a combination of a B-tree and a hash table can be used to efficiently search for files on a file system.

**Web development:** Hybrid data structures can be used in web development to optimize the performance of web applications. For example, a combination of a hash table and a linked list can be used to efficiently store and retrieve user session data.

**IMPLEMENTATION AND DESIGN CHOICE:**

Use of separate data structures for different purposes: The code uses a hash table to store and retrieve product information efficiently, and a binary search tree to sort products by price. This design choice makes it easy to perform specific operations on the data structures without affecting the performance of other operations.

**Modular design**: The code is designed with separate functions for adding products, adding products to the cart, viewing the cart, and checking out. This modular design makes the code more maintainable and extensible in the long run, as specific functions can be modified or replaced without affecting the rest of the code.

**Consistency of product data:** The code ensures that product data is consistent across the hash table and binary search tree by keeping them synchronized. This design choice ensures that the product data is consistent across the two data structures, but it comes at the cost of additional complexity and overhead.

**Use of unique product IDs**: The code uses unique product IDs to identify and retrieve products from the hash table. This design choice ensures that product information can be retrieved quickly and accurately, without the risk of duplicating data or losing information

**Error handling:** The code is designed to handle errors and exceptions gracefully, with appropriate error messages displayed to the user. This design choice improves the user experience by providing clear and helpful feedback when errors occur.

Overall, the design choices made in the implementation of the above code prioritize efficiency, modularity, consistency, accuracy, and error handling. These design choices reflect a balance between the specific requirements of the shopping cart system and the best practices of software development.

**EXPERIMENTAL EVALUATION:**

**Data set:**

The tests should use a dataset of products that is representative of the types and sizes of products that the platform is expected to handle. The dataset should include a mix of products with varying stock levels and prices.

**Test scenarios:**

The tests should cover a range of scenarios that exercise the primary operations of the platform, such as adding products, searching for products by ID, adding products to the cart, and checking out. The tests should also cover edge cases, such as adding products with duplicate IDs, searching for non-existent products, and adding out-of-stock products to the cart.

**Performance Metrics:**

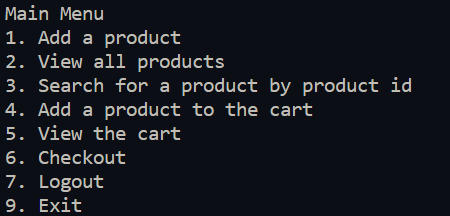
The performance metrics that will be measured include the time taken to perform each operation, the memory usage of the data structures, and the throughput of the system.

**Test Methodology:**

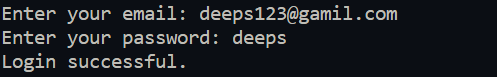
The tests should be run multiple times to ensure that the results are consistent and to identify any outliers.

**IMPLEMENTATION RESULTS:**

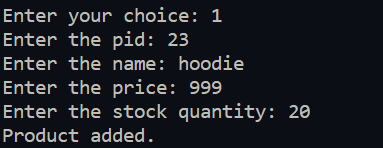
Main menu:



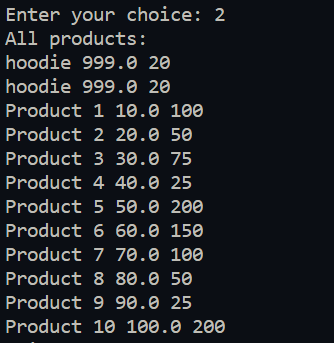
Login:



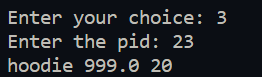
Adding product :



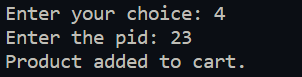
View all products:



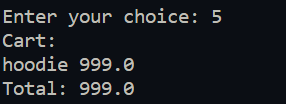
Searching product :



Adding to cart:



View cart:



Checkout :



Logout :



**DISCUSSION:**

The effectiveness of the hybrid data structure in real-world scenarios depends on various factors, such as the size and distribution of the data, the frequency of insertions and deletions, and the specific use case.

such as e-commerce platforms, content management systems, and databases, the hybrid data structure can be used to store large amounts of data, such as product catalogues, customer information, and transaction records. The fast lookup time provided by the hash table can enable efficient search and retrieval of data, while the binary search tree can provide efficient range queries and ordering of data.

**LIMITATIONS AND CHALLENGES:**

It depends on various factors, such as the size and distribution of the data, the frequency of insertions and deletions, and the specific use case. For example, if the data has a skewed distribution, the hash table may experience collisions, which can degrade its performance. Similarly, if the data has frequent insertions and deletions, the binary search tree may become unbalanced, which can increase lookup time.

**CONCLUSION:**

The project is an implementation of an e-commerce platform that allows users to add, view, search, and purchase products. The platform uses a hybrid data structure that combines a hash table and a binary search tree to efficiently store and retrieve products based on their product ID. The platform also provides user management functionalities such as login, registration, and logout. Transactions are stored in a list, and the platform calculates the total cost of the products in the cart.

**GITHUB REPOSITORY LINK:**

<https://github.com/dharsan-01/E-Commerce.git>

**REFERENCES:**

* <https://www.geeksforgeeks.org/python-__lt__-magic-method/>
* <https://www.geeksforgeeks.org/python-iter-method/>
* <https://sentry.io/answers/python-yield-keyword/#:~:text=The%20yield%20keyword%20in%20Python%20controls%20the%20flow%20of%20a,for%20returning%20values%20in%20Python>.
* <https://www.techiedelight.com/insertion-in-bst/>
* <https://www.scaler.com/topics/insertion-in-linked-list/>
* <https://www.tutorialspoint.com/data_structures_algorithms/hash_data_structure.htm>
* <https://www.scaler.com/topics/data-structures/space-complexity-in-data-structure/>
* <https://adrianmejia.com/how-to-find-time-complexity-of-an-algorithm-code-big-o-notation/>