Deliverable 2: Proof of Concept Implementation

Bereket Gebremariam, Sachin Karki

Dr. Satish Penmatsa

2025 Fall - Algorithms and Data Structures (MSCS-532-M80)

University of the Cumberland

Hash Table (Dictionary) Proof-of-Concept Implementation

This section provides a clear explanation of how the hash table is used within the TaskScheduler class to store and manage task metadata efficiently. The hash table functions as the primary repository for task information and plays a critical role in ensuring fast access and updates to task data.

The core purpose of the hash table is to support three fundamental operations: inserting new tasks, retrieving task information, and removing completed tasks, which form the basis of efficient searching and sorting in computer science (Knuth, 1998). These operations are implemented through the add\_task, find\_task, and complete\_task methods in the TaskScheduler class. Because the hash table is implemented using Python’s dictionary data structure, each of these operations executes in constant time, or O(1), on average. This performance consistency is crucial, especially when managing numerous tasks in real-time environments. However, in scenarios demanding strict guarantees, advanced techniques like Cuckoo Hashing are sometimes employed to achieve a worst-case $O(1)$ lookup time (Pagh & Rodler, 2004).

In the hash table, the unique task\_id is used as the key, while the value associated with each key is a dictionary containing metadata about the task, including the deadline, urgency level, and a brief description. This design ensures that the stored metadata is well-structured, easy to understand, and simple to extend should additional attributes need to be included later.

To demonstrate the effectiveness of the hash table, a test script was used to showcase insertion, lookup, and deletion operations, including the handling of edge cases. For example, inserting tasks such as T101 and T102 confirms that data is stored correctly. A lookup for T102 returns its details instantly, highlighting the speed of the hash table. Attempts to access a non-existent task, such as T999, confirm that appropriate error messages are displayed. Additionally, deleting a task and attempting to delete it again verifies that duplicate deletions are managed gracefully.

A computer screen shot of text

AI-generated content may be incorrect.  
Figure Hash Table

The hash table also plays a vital role in the overall design of the task scheduling system, particularly when integrating future enhancements. As the system expands to include components such as a priority queue implemented using a min-heap, a common technique in high-speed scheduling systems (Ioannou & Katevenis, 2006), the hash table will serve as the single source of truth for determining whether task entries in other structures are valid or outdated. This capability is essential for supporting a lazy deletion strategy, where tasks in other structures may not always be immediately removed.

From a development perspective, several important considerations guided this implementation. Ensuring data consistency required designating the hash table as the authoritative source for all task metadata. Structuring the stored metadata as a descriptive dictionary made the implementation more intuitive while supporting future growth. Error handling was also incorporated to help maintain system stability and provide meaningful feedback to users.

To complete the full system implementation, the next steps include integrating the hash table with a min-heap to support priority-based scheduling, implementing lazy deletion logic to manage outdated records efficiently, constructing a multi-attribute priority key to ensure accurate task ordering, and developing a command-line interface to support user interaction. By following these steps, the task management system will evolve into a robust, efficient, and user-friendly application.

References

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

Ioannou, A., & Katevenis, M. (2006). Pipelined heap (priority queue) management for advanced scheduling in high-speed networks. *IEEE/ACM Transactions on Networking*, *14*(3), 540–553.

Knuth, D. E. (1998). *The art of computer programming, volume 3: Sorting and searching* (2nd ed.). Addison-Wesley Professional.

Pagh, R., & Rodler, F. F. (2004). Cuckoo hashing. *Journal of Algorithms*, *51*(2), 122–144.