**SE CS BATCH 2023**

**FALL SEMESTER 2024**

**PROBLEM DESCRIPTION**

Design a data structure in Python that follows the constraints of a Least Recently Used (LRU) cache and find its time and space complexities

**Flow of the Project (Data Structures & Algorithms)**

To solve the LRU Cache problem, we use the following data structures and algorithms:

#### **Data Structures:**

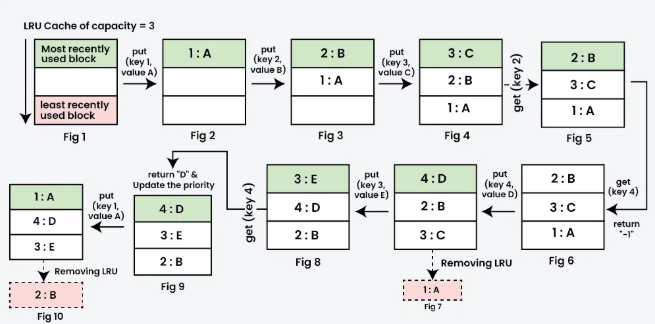
1. **Doubly Linked List:**
   * This is used to maintain the order of keys based on their usage. The most recently used item is placed at the tail, and the least recently used item is placed at the head.
   * Each node in the list contains a key-value pair and pointers to both the previous and next nodes**.**
2. **HashMap:**
   * A dictionary (hash map) is used to store key-value pairs, where the key is mapped to its corresponding node in the doubly linked list. This allows O(1) access to any key.

#### **Algorithms:**

1. **Get Operation:**
   * If the key exists in the cache (i.e., in the hash map), it is moved to the tail of the doubly linked list to mark it as the most recently used.
   * If the key does not exist, the method returns -**1**, indicating a cache miss.
2. **Put Operation:**
   * If the key already exists, the value of the key is updated
   * If the key does not exist and the cache has reached its capacity, the least recently used item (node at the head) is removed and the new node is inserted at the tail.
3. **Remove and Add Operations:**
   * The remove operation takes a node and removes it from the doubly linked list.
   * The add operation inserts a node just before the tail to mark it as the most recently used.

#### **Flow:**

1. When the cache is initialized, two dummy nodes (head and tail) are created to simplify the insertion and deletion operations.
2. Each time a key is accessed, it is moved to the tail of the doubly linked list.
3. When the cache reaches its capacity, the node at the head (the least recently used) is removed.



**Most Challenging Part**

The most challenging part of the project was choosing the right data structures and managing the doubly linked list efficiently to ensure quick operations for adding, removing, and moving items ( **all in constant time, O(1)** ). We had to keep the least recently used item at the head and the most recent at the tail, which meant carefully managing the pointers. It was also important to make sure the dictionary (hash map) and the linked list stayed in sync during all operations, especially when removing items.

**New Learning**

I learned about the LRU eviction strategy, which helps remove the least recently used item in a cache. This is important in systems with limited memory to make sure the most frequently used data stays available.