LRU Cache Documentation

Overview

The Least Recently Used (LRU) Cache is a data structure that maintains a limited number of items and evicts the least recently used items when the cache exceeds its capacity. The operations for accessing and updating cache items are designed to run in constant time, O(1), to ensure efficient performance.

Class: LRUCache

Description

The LRUCache class implements a cache that follows the Least Recently Used (LRU) eviction policy. It provides efficient methods for retrieving and updating items while ensuring that the cache does not exceed a specified capacity.

Initialization

LRUCache(int capacity)

- **Description:** Initializes the LRU cache with a given capacity.

- **Parameters:**
  - *capacity (int):* The maximum number of items that the cache can hold. It must be a positive integer.

- **Example:** lRUCache = LRUCache(2)

Methods

Int get(int key)

- **Description:** Retrieves the value associated with the specified key from the cache if it exists. If the key does not exist, it returns -1. This operation updates the key to be the most recently used.

- **Parameters:**
  - *key (int):* The key whose associated value is to be retrieved.
  - *Returns:* The value associated with the key if it is present in the cache; otherwise, -1.

- **Example:** value = lRUCache.get(1) # Retrieves the value for key 1 or returns -1 if the key does not exist

Void put(int key, int value)

- **Description:** Adds or updates the value associated with the specified key in the cache. If the key already exists, its value is updated and the key is moved to the most recently used position. If the key does not exist and the cache is at full capacity, the least recently used item is evicted to make space for the new key-value pair.

- **Parameters:**
  - *key (int):* The key to be added or updated.
  - *value (int):* The value to be associated with the key.
  - *Returns:* None

- **Example:**

➢ lRUCache.put(1, 1) # Adds or updates the key-value pair (1, 1)

➢ lRUCache.put(2, 2) # Adds or updates the key-value pair (2, 2)

Example Usage

Create an LRUCache with a capacity of 2

lRUCache = LRUCache(2)

Add key-value pairs to the cache

lRUCache.put(1, 1) # Cache is {1=1}

lRUCache.put(2, 2) # Cache is {1=1, 2=2}

Access the value associated with key 1

print(lRUCache.get(1)) # Output: 1

Add a new key-value pair, causing the least recently used key (2) to be evicted

lRUCache.put(3, 3) # Cache is {1=1, 3=3}

Access the value associated with key 2 (should be -1 as it was evicted)

print(lRUCache.get(2)) # Output: -1

Add another key-value pair, evicting the least recently used key (1)

lRUCache.put(4, 4) # Cache is {4=4, 3=3}

Access values for keys 1, 3, and 4

print(lRUCache.get(1)) # Output: -1 (evicted)

print(lRUCache.get(3)) # Output: 3

print(lRUCache.get(4)) # Output: 4

Constraints

- 1 <= capacity <= 3000
- 0 <= key <= 10^4
- 0 <= value <= 10^5
- At most 2 * 10^5 calls will be made to get and put.

Implementation Details

- **Data Structures Used:**

➢ *Hash Map:* For quick lookups of key-value pairs.

- **Doubly Linked List:** To maintain the order of elements based on their usage, allowing O(1) operations for adding and removing nodes.

- **Operations:**

➢ *Get Operation:* Moves the accessed node to the front of the list to mark it as most recently used.

➢ *Put Operation:* Adds or updates the node in the cache. If the cache exceeds its capacity, it evicts the least recently used node (the node before the dummy tail).