**📘 README.md**

**Q1: LRU Cache (Least Recently Used)**

**Problem Statement:**

Implement an **LRU Cache** with O(1) time for get(key) and put(key, value) operations using:

* **Doubly Linked List** (to maintain order of usage)
* **Hash Map** (for fast access)

**📂 Files:**

* a1.py

**🧪 Sample Test:**

cpp

CopyEdit

LRUCache lru(2);

lru.put(1, 1);

lru.put(2, 2);

lru.get(1); // returns 1

lru.put(3, 3); // evicts key 2

lru.get(2); // returns -1 (not found)

lru.put(4, 4); // evicts key 1

lru.get(1); // returns -1 (not found)

lru.get(3); // returns 3

lru.get(4); // returns 4

**✅ Output:**

diff

CopyEdit

1

-1

-1

3

4

**Q2: Custom HashMap Implementation**

**Problem Statement:**

Design a **custom HashMap** class that supports:

* put(key, value) – Insert or update value by key
* get(key) – Return value or -1 if not found
* remove(key) – Delete key from map

Do **not use** unordered\_map, map, or other built-in hash tables.

**📂 Files:**

* a2.py

**⚙️ Implementation Details:**

* Use **separate chaining** (array of linked lists)
* Hash function: key % bucketSize for distribution
* Supports up to 10^6 keys and 10^5 operations

**🧪 Sample Test:**

cpp

CopyEdit

MyHashMap obj;

obj.put(1, 10);

obj.put(2, 20);

obj.get(1); // returns 10

obj.get(3); // returns -1

obj.put(2, 30); // update key 2

obj.get(2); // returns 30

obj.remove(2);

obj.get(2); // returns -1

**✅ Output:**

diff

CopyEdit

10

-1

30

-1