Here are **bullet-point notes** for designing an **LRU (Least Recently Used) Cache**, a key system for optimizing memory and lookup efficiency:

**🧩 1. Problem Statement**

* Design a cache that:
  + Stores up to **N** items.
  + On cache **miss**, loads the item.
  + On cache **hit**, returns quickly.
  + **Evicts** the **least recently used** item when full.

**📎 2. Requirements**

**✅ Functional**

* get(key) → Return value, update usage.
* put(key, value) → Add or update, evict LRU if needed.

**❌ Non-Functional**

* **O(1)** time complexity for both get and put.
* Thread-safe (if used in multi-threaded environments).

**🧠 3. Core Design Idea**

* Combine:
  + **Doubly Linked List**: Maintains order of usage (most recent at head, least at tail).
  + **Hash Map**: For O(1) access by key.

**📦 4. Data Structures**

* HashMap<key, Node\*>:
  + Fast access to the node in the list.
* Doubly Linked List<Node>:
  + Node contains key, value.
  + Most recently used node at **head**.
  + Least recently used node at **tail**.

**🔄 5. Operations**

**get(key)**

* If key exists:
  + Move node to front of list.
  + Return value.
* Else return -1 (or null).

**put(key, value)**

* If key exists:
  + Update value.
  + Move node to front.
* Else:
  + If cache is full:
    - Remove node from tail (LRU).
    - Delete from map.
  + Add new node at head.
  + Insert in map.

**⚙️ 6. Time and Space Complexity**

* **Time**:
  + get: O(1)
  + put: O(1)
* **Space**: O(capacity)

**🧵 7. Concurrency (Optional)**

* Use mutex/lock if in multi-threaded context.
* Consider read-write locks for get vs put.

**🧪 8. Example Walkthrough**

text

CopyEdit

capacity = 2

put(1,1) → cache: [1]

put(2,2) → cache: [2,1]

get(1) → returns 1, cache: [1,2]

put(3,3) → evicts 2, cache: [3,1]

**🧰 9. Implementation Tips**

* Create dummy head and tail nodes to simplify insert/remove logic.
* Link nodes using prev and next pointers.
* Always insert at head, remove from tail.

**📦 10. Real-World Usage**

* CPU/L1-L3 memory caching.
* Web browsers (page caching).
* Database query caching.
* Content delivery networks (CDNs).