LRU Cache

For this problem we need to design a data structure with O(1) get and put. The best approach combines:

- Hash Map (unordered_map in C++) → for O(1) access by key.
- Doubly Linked List → to maintain the order of usage (most recently used at tail).

Approach:

1. Data structures:
    - Doubly Linked List:
        - Each node stores (key, value).
        - Most recently used node is near the head.
        - Least recently used node is near the tail.
    - Hash Map:
        - Maps key → node pointer in the linked list.
2. Operations:
    - get (key):
        - If key exists:
            - Move the node to the head (most recently used).
            - Return the value.
        - Else return -1
    - put (key, value):
        - If key exists:
            - Update value and move node to head.
        - Else:
            - Create new node, insert at head.
            - If size exceeds capacity, remove node at tail (least recently used).

Implementation Details:

- Use a dummy head and dummy tail for easier node management.
- Functions:

- addNode(node): → insert node right after head.
- removeNode(node): → unlink node from list.
- moveToHead(node): → remove and re-insert at head.
- popTail() → remove last node (before tail).

Complexity:

- Time Complexity: O(1) for get and put
- Space Complexity: O(capacity) for storing hash map and linked list nodes