DAILY DSA | DAY-12 | Linked lists – Practice|
-GOPALKRISHNA A

Today, we will put the linked lists concepts into practice.

Problem to solve: LRU Cache
The program developed by you is running slowly because it's accessing data from the disk over and over again. To improve the performance, you want to build a simple key-value store to cache this data in memory, but you also want to limit the amount of memory used. You decide to build a caching system that only keeps the N most recently used items - also known as the **least recently used (LRU) cache**

Write a class LRUCache(n) that accepts a size limit n. it should support a set(key, value) method for inserting or updating items and get(key) method for retrieving items.

Need to implement a solution where both of these methods run in O(1) time

```
class LRUCache(n):
  set(key, value)
  get(key)
```

Example:
To make room for new items, the least recently used item should be removed. An item is 'used' whenever it is set, retrieved, or updated.

```
cache = LRUCache(2) # Limit of 2 items

cache.set('user1', 'Alex')
cache.set('user2', 'Brian')
cache.set('user3', 'Chris')
```

Here user1 is empty because it was the least recently used and thus removed to make room for user3

```
cache.get('user1') # => None
cache.get('user2') # => 'Brian'
cache.get('user3') # => 'Chris'
```

Hints:

1. To get started, think about data structure(s) that can be used to solve this problem. Which one is best for a key-value store? which can help to store items in order of recency
2. Need to find and remove the least recent item in O(1) time. An array won't work because moving items around takes O(n) time
3. Can use doubly linked lists to store the items in the correct order. When an item is used, just move its node to the front of the list. The least recently used node will always end up at the end of the list
4. To look up items without traversing the entire list, we can also add a hash table that maps from the key to the item's node

More practice problems:
https://www.geeksforgeeks.org/data-structures/linked-list/