Prompt

Difficulty: **Category:** Linked Lists

LRU Cache

Implement an `LRUCache` class for a Least Recently Used (LRU) cache. The class should support:

- Inserting key-value pairs with the `insertKeyValuePair` method.
- Retrieving a key's value with the `getValueFromKey` method.
- Retrieving the most recently used (the most recently inserted or retrieved) key with the `getMostRecentKey` method.

Each of these methods should run in constant time.

Additionally, the `LRUCache` class should store a `maxSize` property set to the size of the cache, which is passed in as an argument during instantiation. This size represents the maximum number of key-value pairs that the cache can store at once. If a key-value pair is inserted in the cache when it has reached maximum capacity, the least recently used key-value pair should be evicted from the cache and no longer retrievable; the newly added key-value pair should effectively replace it.

Note that inserting a key-value pair with an already existing key should simply replace the key's value in the cache with the new value and shouldn't evict a key-value pair if the cache is full. Lastly, attempting to retrieve a value from a key that isn't in the cache should return `None` / `null`.

Sample Usage

```
// All operations below are performed sequentially.
LRUCache(3): - // instantiate an LRUCache of size 3
insertKeyValuePair("b", 2): -
insertKeyValuePair("a", 1): -
insertKeyValuePair("c", 3): -
getMostRecentKey(): "c" // "c" was the most recently inserted key
getValueFromKey("a"): 1
getMostRecentKey(): "a" // "a" was the most recently retrieved key
insertKeyValuePair("d", 4): - // the cache had 3 entries; the least recently
getValueFromKey("b"): None // "b" was evicted in the previous operation
insertKeyValuePair("a", 5): - // "a" already exists in the cache so its valu
getValueFromKey("a"): 5
```

Hints

Hint 1

What data structure could allow you to insert, retrieve, and evict resources as fast as possible, all the while keeping track of the least recently accessed resource - essentially keeping track of the order of the resources? A hash table would allow you to insert and retrieve resources fast, but it wouldn't allow you to keep track of their order. An array would let you keep track of their order, but it wouldn't let you access elements fast; it also wouldn't allow you to move an element from one position to another in constant time, which you would need to do to make a newly-accessed key / value pair the most recent one upon retrieval of a key's value. A linked list would allow you to keep track of elements' order and to move them seamlessly (if you knew their position), but it wouldn't allow you to access them easily without knowing their position beforehand. Could a heap help? What about a BST or a trie? Would any other data structures work?

Hint 2

Could you use multiple data structures to make your LRU Cache's functionality fast and efficient? Could you store keys in one data structure, for instance, and values in an auxiliary data structure? What should these data structures be in order for all of the LRU Cache's methods to run in constant time?

Hint 3

Try storing keys in a hash table and mapping them to nodes in a doubly linked list containing the keys' corresponding values (perhaps the nodes would also have to store the keys themselves). With these two data structures, you could access any key / value pair very easily via the hash table, and you could also effortlessly move nodes in the linked list so as to keep track of the most recent and least recent key / value pairs. The linked list would also allow you to keep track of the entire order of the key / value pairs, thus allowing you to perpetually update the least recent key / value pairs after evictions.

Optimal Space & Time Complexity

(all 3 methods) O(1) time | O(1) space