Design and implement a data structure for [Least Recently Used (LRU) cache](https://en.wikipedia.org/wiki/Cache_replacement_policies#LRU). It should support the following operations: get and put.

get(key) - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.  
put(key, value) - Set or insert the value if the key is not already present. When the cache reached its capacity, it should invalidate the least recently used item before inserting a new item.

**Follow up:**  
Could you do both operations in **O(1)** time complexity?

Solution: For Implementing everything to be O(1), we need a data structure which is O(1) in insertion and O(1) in search. First things which come to mind is that linked lists are O(1) insertion operations if items are inserted at head / talk. Then we have Map that does O(1) lookups. So using these two data structures, we can build an all O(1) operations. The most recently used items are at the head and the least recently used items will be at the tail end. We’ll keep track of nodes in the list with the map which maps from keys to nodes. When the linked list reaches the capacity, we’ll remove one item at a time from the tail and then insert the new keys. If the key is already present, then we’ll remove it from the current position in the linked list and move it to the head of the list.