Problem #146: LRU Cache

<https://leetcode.com/problems/lru-cache/>

from collections import OrderedDict

class LRUCache:

def \_\_init\_\_(self, capacity: int):

self.capacity = capacity

self.cache = OrderedDict()

def get(self, key: int) -> int:

if key not in self.cache.keys():

return(-1)

#else:

value = self.cache[key] # get the value of the key

self.cache.pop(key) # remove the entry from the cache

self.cache[key] = value # append the key and value to the cache so that it goes at the end

return(value)

def put(self, key: int, value: int) -> None:

if key in self.cache: # if key is already in the cache we need to update its value

self.cache.pop(key) # we remove the entry for the key in the cache

else:

if len(self.cache) >= self.capacity: # length of cache is exceeding capacity

self.cache.popitem(last = False) # Remove the first entry from the cache

self.cache[key] = value # Add the new item to the cache after making space if needed

# Your LRUCache object will be instantiated and called as such:

# obj = LRUCache(capacity)

# param\_1 = obj.get(key)

# obj.put(key,value)