LRU Cache

1. 核心逻辑是用Ordered Dictionary (implemented using double linked list and a hashmap)

- One advantage of using double linked list is that node can remove itself without other reference, and it takes constant take to add and remove nodes from head or tail.

```
class DLinkedNode():
    def __init__(self):
        self.key = 0
        self.value = 0
        self.prev = None
        self.next = None

class LRUCache():
    def _add_node(self, node):
        """
        Always add the new node right after head.
        """
        node.prev = self.head
        node.next = self.head.next

        self.head.next.prev = node
        self.head.next = node

    def _remove_node(self, node):
        """
        Remove an existing node from the linked list.
        """
        prev = node.prev
        new = node.next

        prev.next = new
        new.prev = prev
```

```
    def _move_to_head(self, node):
        """
        Move certain node in between to the head.
        """
        self._remove_node(node)
        self._add_node(node)

    def _pop_tail(self):
        """
        Pop the current tail.
        """
        res = self.tail.prev
        self._remove_node(res)
        return res

    def __init__(self, capacity):
        """
        :type capacity: int
        """
        self.cache = {}
        self.size = 0
        self.capacity = capacity
        self.head, self.tail = DLinkedNode(), DLinkedNode()

        self.head.next = self.tail
        self.tail.prev = self.head


    def get(self, key):
        """
        :type key: int
        :rtype: int
        """
        node = self.cache.get(key, None)
        if not node:
            return -1

        # move the accessed node to the head;
        self._move_to_head(node)

        return node.value

    def put(self, key, value):
        """
        :type key: int
        :type value: int
        :rtype: void
        """
        node = self.cache.get(key)

        if not node:
            newNode = DLinkedNode()
            newNode.key = key
```

```
            newNode.value = value

            self.cache[key] = newNode
            self._add_node(newNode)

            self.size += 1

            if self.size > self.capacity:
                # pop the tail
                tail = self._pop_tail()
                del self.cache[tail.key]
                self.size -= 1
        else:
            # update the value.
            node.value = value
            self._move_to_head(node)
```