Solution 1

```python
# Copyright © 2020 AlgoExpert, LLC. All rights reserved.

class LRUCache:
    def __init__(self, maxSize):
        self.cache = {}
        self.maxSize = maxSize or 1
        self.currentSize = 0
        self.listOfMostRecent = DoublyLinkedList()

    # O(1) time | O(1) space
    def insertKeyValuePair(self, key, value):
        if key not in self.cache:
            if self.currentSize == self.maxSize:
                self.evictLeastRecent()
            else:
                self.currentSize += 1
            self.cache[key] = DoublyLinkedListNode(key, value)
        else:
            self.replaceKey(key, value)
        self.updateMostRecent(self.cache[key])

    # O(1) time | O(1) space
    def getValueFromKey(self, key):
        if key not in self.cache:
            return None
        self.updateMostRecent(self.cache[key])
        return self.cache[key].value

    # O(1) time | O(1) space
    def getMostRecentKey(self):
        return self.listOfMostRecent.head.key

    def evictLeastRecent(self):
        keyToRemove = self.listOfMostRecent.tail.key
        self.listOfMostRecent.removeTail()
        del self.cache[keyToRemove]

    def updateMostRecent(self, node):
        self.listOfMostRecent.setHeadTo(node)

    def replaceKey(self, key, value):
        if key not in self.cache:
            raise Exception("The provided key isn't in the cache!")
        self.cache[key].value = value


class DoublyLinkedList:
    def __init__(self):
        self.head = None
        self.tail = None

    def setHeadTo(self, node):
        if self.head == node:
            return
        elif self.head is None:
            self.head = node
            self.tail = node
        elif self.head == self.tail:
            self.tail.prev = node
            self.head = node
            self.head.next = self.tail
        else:
            if self.tail == node:
                self.removeTail()
            node.removeBindings()
            self.head.prev = node
            node.next = self.head
            self.head = node

    def removeTail(self):
        if self.tail is None:
            return
        if self.tail == self.head:
            self.head = None
            self.tail = None
            return
        self.tail = self.tail.prev
        self.tail.next = None


class DoublyLinkedListNode:
    def __init__(self, key, value):
        self.key = key
        self.value = value
        self.prev = None
        self.next = None

    def removeBindings(self):
        if self.prev is not None:
            self.prev.next = self.next
        if self.next is not None:
            self.next.prev = self.prev
        self.prev = None
        self.next = None

```