Solution 1

```
// Copyright © 2020 AlgoExpert, LLC. All rights reserved.

class LRUCache {
  constructor(maxSize) {
    this.cache = {};
    this.maxSize = maxSize || 1;
    this.currentSize = 0;
    this.listOfMostRecent = new DoublyLinkedList();
  }

  // O(1) time | O(1) space
  insertKeyValuePair(key, value) {
    if (!(key in this.cache)) {
      if (this.currentSize === this.maxSize) {
        this.evictLeastRecent();
      } else {
        this.currentSize++;
      }
      this.cache[key] = new DoublyLinkedListNode(key, value);
    } else {
      this.replaceKey(key, value);
    }
    this.updateMostRecent(this.cache[key]);
  }

  // O(1) time | O(1) space
  getValueFromKey(key) {
    if (!(key in this.cache)) return null;
    this.updateMostRecent(this.cache[key]);
    return this.cache[key].value;
  }

  // O(1) time | O(1) space
  getMostRecentKey() {
    return this.listOfMostRecent.head.key;
  }

  evictLeastRecent() {
    const keyToRemove = this.listOfMostRecent.tail.key;
    this.listOfMostRecent.removeTail();
    delete this.cache[keyToRemove];
  }

  updateMostRecent(node) {
    this.listOfMostRecent.setHeadTo(node);
  }

  replaceKey(key, value) {
    if (!(key in this.cache)) {
      throw new Error("The provided key isn't in the cache!");
    }
    this.cache[key].value = value;
  }
}

class DoublyLinkedList {
  constructor() {
    this.head = null;
    this.tail = null;
  }

  setHeadTo(node) {
    if (this.head === node) {
      return;
    } else if (this.head === null) {
      this.head = node;
      this.tail = node;
    } else if (this.head === this.tail) {
      this.tail.prev = node;
      this.head = node;
      this.head.next = this.tail;
    } else {
      if (this.tail === node) this.removeTail();
      node.removeBindings();
      this.head.prev = node;
      node.next = this.head;
      this.head = node;
    }
  }

  removeTail() {
    if (this.tail === null) return;
    if (this.tail === this.head) {
      this.head = null;
      this.tail = null;
      return;
    }
    this.tail = this.tail.prev;
    this.tail.next = null;
  }
}

class DoublyLinkedListNode {
  constructor(key, value) {
    this.key = key;
    this.value = value;
    this.prev = null;
    this.next = null;
  }

  removeBindings() {
    if (this.prev !== null) {
      this.prev.next = this.next;
    }
    if (this.next !== null) {
      this.next.prev = this.prev;
    }
    this.prev = null;
    this.next = null;
  }
}

exports.LRUCache = LRUCache;
```