Solution 1

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

package main

type LRUCache struct {
  index            map[string]*DoublyLinkedListNode
  maxSize          int
  currentSize      int
  listOfMostRecent *DoublyLinkedList
}

func NewLRUCache(size int) *LRUCache {
  lru := &LRUCache{
    index:            map[string]*DoublyLinkedListNode{},
    maxSize:          size,
    currentSize:      0,
    listOfMostRecent: &DoublyLinkedList{},
  }
  if lru.maxSize < 1 {
    lru.maxSize = 1
  }
  return lru
}

// O(1) time | O(1) space
func (cache *LRUCache) InsertKeyValuePair(key string, value int) {
  if _, found := cache.index[key]; !found {
    if cache.currentSize == cache.maxSize {
      cache.evictLeastRecent()
    } else {
      cache.currentSize += 1
    }
    cache.index[key] = &DoublyLinkedListNode{
      key:   key,
      value: value,
    }
  } else {
    cache.replaceKey(key, value)
  }
  cache.updateMostRecent(cache.index[key])
}

// O(1) time | O(1) space
func (cache *LRUCache) GetValueFromKey(key string) (int, bool) {
  if node, found := cache.index[key]; !found {
    return 0, false
  } else {
    cache.updateMostRecent(node)
    return node.value, true
  }
}

// O(1) time | O(1) space
func (cache *LRUCache) GetMostRecentKey() (string, bool) {
  if cache.listOfMostRecent.head == nil {
    return "", false
  }
  return cache.listOfMostRecent.head.key, true
}

func (cache *LRUCache) evictLeastRecent() {
  key := cache.listOfMostRecent.tail.key
  cache.listOfMostRecent.removeTail()
  delete(cache.index, key)
}

func (cache *LRUCache) updateMostRecent(node *DoublyLinkedListNode) {
  cache.listOfMostRecent.setHeadTo(node)
}

func (cache *LRUCache) replaceKey(key string, value int) {
  if node, found := cache.index[key]; !found {
    panic("The provided key isn't in the cache!")
  } else {
    node.value = value
  }
}

type DoublyLinkedList struct {
  head *DoublyLinkedListNode
  tail *DoublyLinkedListNode
}

func (list *DoublyLinkedList) setHeadTo(node *DoublyLinkedListNode) {
  if list.head == node {
    return
  }
  if list.head == nil {
    list.head, list.tail = node, node
    return
  }
  if list.head == list.tail {
    list.tail.prev = node
    list.head = node
    list.head.next = list.tail
    return
  }
  if list.tail == node {
    list.removeTail()
  }
  node.removeBindings()
  list.head.prev = node
  node.next = list.head
  list.head = node
}

func (list *DoublyLinkedList) removeTail() {
  if list.tail == nil {
    return
  }
  if list.tail == list.head {
    list.head, list.tail = nil, nil
    return
  }
  list.tail = list.tail.prev
```

```
  list.tail.next = nil
}

type DoublyLinkedListNode struct {
  key   string
  value int
  prev  *DoublyLinkedListNode
  next  *DoublyLinkedListNode
}

func (node *DoublyLinkedListNode) removeBindings() {
  if node.prev != nil {
    node.prev.next = node.next
  }
  if node.next != nil {
    node.next.prev = node.prev
  }
  node.prev, node.next = nil, nil
}

```