Solution 1

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

class Program {
    class LRUCache {
        var maxSize: Int
        var currentSize = 0
        var cache = [String: DoublyLinkedListNode]()
        var listOfMostRecent = DoublyLinkedList()

        init(maxSize: Int) {
            self.maxSize = maxSize
        }

        // O(1) time | O(1) space
        func insertKeyValuePair(_ key: String, _ value: Int) {
            if !cache.keys.contains(key) {
                if currentSize == maxSize {
                    evictLeastRecent()
                } else {
                    currentSize += 1
                }

                cache[key] = DoublyLinkedListNode(key, value)
            } else if let existingNode = cache[key] {
                existingNode.value = value
            }

            if let node = cache[key] {
                updateMostRecent(node)
            }
        }

        func evictLeastRecent() {
            if let key = listOfMostRecent.tail?.key {
                listOfMostRecent.removeTail()
                cache[key] = nil
            }
        }

        func updateMostRecent(_ node: DoublyLinkedListNode) {
            listOfMostRecent.setHeadTo(node)
        }

        // O(1) time | O(1) space
        func getValueFromKey(_ key: String) -> Int? {
            if let existingNode = cache[key] {
                updateMostRecent(existingNode)
                return existingNode.value
            } else {
                return nil
            }
        }

        // O(1) time | O(1) space
        func getMostRecentKey() -> String? {
            return listOfMostRecent.head?.key
        }
    }

    class DoublyLinkedListNode {
        let key: String
        var value: Int
        var previous: DoublyLinkedListNode?
        var next: DoublyLinkedListNode?

        init(_ key: String, _ value: Int) {
            self.key = key
            self.value = value
            previous = nil
            next = nil
        }

        func removeBindings() {
            if let previous = previous {
                previous.next = next
            }

            if let next = next {
                next.previous = previous
            }

            previous = nil
            next = nil
        }
    }

    class DoublyLinkedList {
        var head: DoublyLinkedListNode?
        var tail: DoublyLinkedListNode?

        init() {
            head = nil
            tail = nil
        }

        func setHeadTo(_ node: DoublyLinkedListNode) {
            if head === node {
                return
            } else if head === nil {
                head = node
                tail = node
            } else if head === tail {
                tail?.previous = node
                head = node
                head?.next = tail
            } else {
                if tail === node {
                    removeTail()
                }

                node.removeBindings()
                head?.previous = node
                node.next = head
                head = node
            }
```

```
        }

        func removeTail() {
            if tail === nil {
                return
            }

            if head === tail {
                head = nil
                tail = nil
                return
            }

            tail = tail?.previous
            tail?.next = nil
        }
    }
}

```