

Solution 1

```
1 // Copyright © 2020 AlgoExpert, LLC. All rights reserved.
2
3 class Program {
4     class LRUCache {
5         var maxSize: Int
6         var currentSize = 0
7         var cache = [String: DoublyLinkedListNode]()
8         var listOfMostRecent = DoublyLinkedList()
9
10        init(maxSize: Int) {
11            self.maxSize = maxSize
12        }
13
14        // O(1) time | O(1) space
15        func insertKeyValuePair(_ key: String, _ value: Int) {
16            if !cache.keys.contains(key) {
17                if currentSize == maxSize {
18                    evictLeastRecent()
19                } else {
20                    currentSize += 1
21                }
22
23                cache[key] = DoublyLinkedListNode(key, value)
24            } else if let existingNode = cache[key] {
25                existingNode.value = value
26            }
27
28            if let node = cache[key] {
29                updateMostRecent(node)
30            }
31        }
32
33        func evictLeastRecent() {
34            if let key = listOfMostRecent.tail?.key {
35                listOfMostRecent.removeTail()
36                cache[key] = nil
37            }
38        }
39
40        func updateMostRecent(_ node: DoublyLinkedListNode) {
41            listOfMostRecent.setHeadTo(node)
42        }
43
44        // O(1) time | O(1) space
45        func getValueFromKey(_ key: String) -> Int? {
46            if let existingNode = cache[key] {
47                updateMostRecent(existingNode)
48                return existingNode.value
49            } else {
50                return nil
51            }
52        }
53
54        // O(1) time | O(1) space
55        func getMostRecentKey() -> String? {
56            return listOfMostRecent.head?.key
57        }
58    }
59
60    class DoublyLinkedListNode {
61        let key: String
62        var value: Int
63        var previous: DoublyLinkedListNode?
64        var next: DoublyLinkedListNode?
65
66        init(_ key: String, _ value: Int) {
67            self.key = key
68            self.value = value
69            previous = nil
70            next = nil
71        }
72
73        func removeBindings() {
74            if let previous = previous {
75                previous.next = next
76            }
77
78            if let next = next {
79                next.previous = previous
80            }
81
82            previous = nil
83            next = nil
84        }
85    }
86
87    class DoublyLinkedList {
88        var head: DoublyLinkedListNode?
89        var tail: DoublyLinkedListNode?
90
91        init() {
92            head = nil
93            tail = nil
94        }
95
96        func setHeadTo(_ node: DoublyLinkedListNode) {
97            if head == node {
98                return
99            } else if head == nil {
100                head = node
101                tail = node
102            } else if head == tail {
103                tail?.previous = node
104                head = node
105                head?.next = tail
106            } else {
107                if tail == node {
108                    removeTail()
109                }
110
111                node.removeBindings()
112                head?.previous = node
113                node.next = head
114                head = node
115            }
116        }
117    }
118}
```

```
116     }
117
118     func removeTail() {
119         if tail == nil {
120             return
121         }
122
123         if head == tail {
124             head = nil
125             tail = nil
126             return
127         }
128
129         tail = tail?.previous
130         tail?.next = nil
131     }
132 }
133 }
134
```