

Solution 1

```
1 // Copyright © 2020 AlgoExpert, LLC. All rights reserved.
2
3 package main
4
5 type LRUCache struct {
6     index      map[string]*DoublyLinkedListNode
7     maxSize     int
8     currentSize int
9     listOfMostRecent *DoublyLinkedList
10 }
11
12 func NewLRUCache(size int) *LRUCache {
13     lru := &LRUCache{
14         index:      map[string]*DoublyLinkedListNode{},
15         maxSize:     size,
16         currentSize: 0,
17         listOfMostRecent: &DoublyLinkedList{},
18     }
19     if lru.maxSize < 1 {
20         lru.maxSize = 1
21     }
22     return lru
23 }
24
25 // O(1) time | O(1) space
26 func (cache *LRUCache) InsertKeyValuePair(key string, value int) {
27     if _, found := cache.index[key]; !found {
28         if cache.currentSize == cache.maxSize {
29             cache.evictLeastRecent()
30         } else {
31             cache.currentSize += 1
32         }
33         cache.index[key] = &DoublyLinkedListNode{
34             key:   key,
35             value: value,
36         }
37     } else {
38         cache.replaceKey(key, value)
39     }
40     cache.updateMostRecent(cache.index[key])
41 }
42
43 // O(1) time | O(1) space
44 func (cache *LRUCache) GetValueFromKey(key string) (int, bool) {
45     if node, found := cache.index[key]; !found {
46         return 0, false
47     } else {
48         cache.updateMostRecent(node)
49         return node.value, true
50     }
51 }
52
53 // O(1) time | O(1) space
54 func (cache *LRUCache) GetMostRecentKey() (string, bool) {
55     if cache.listOfMostRecent.head == nil {
56         return "", false
57     }
58     return cache.listOfMostRecent.head.key, true
59 }
60
61 func (cache *LRUCache) evictLeastRecent() {
62     key := cache.listOfMostRecent.tail.key
63     cache.listOfMostRecent.removeTail()
64     delete(cache.index, key)
65 }
66
67 func (cache *LRUCache) updateMostRecent(node *DoublyLinkedListNode) {
68     cache.listOfMostRecent.setHeadTo(node)
69 }
70
71 func (cache *LRUCache) replaceKey(key string, value int) {
72     if node, found := cache.index[key]; !found {
73         panic("The provided key isn't in the cache!")
74     } else {
75         node.value = value
76     }
77 }
78
79 type DoublyLinkedList struct {
80     head *DoublyLinkedListNode
81     tail *DoublyLinkedListNode
82 }
83
84 func (list *DoublyLinkedList) setHeadTo(node *DoublyLinkedListNode) {
85     if list.head == node {
86         return
87     }
88     if list.head == nil {
89         list.head, list.tail = node, node
90         return
91     }
92     if list.head == list.tail {
93         list.tail.prev = node
94         list.head = node
95         list.head.next = list.tail
96         return
97     }
98     if list.tail == node {
99         list.removeTail()
100     }
101     node.removeBindings()
102     list.head.prev = node
103     node.next = list.head
104     list.head = node
105 }
106
107 func (list *DoublyLinkedList) removeTail() {
108     if list.tail == nil {
109         return
110     }
111     if list.tail == list.head {
112         list.head, list.tail = nil, nil
113         return
114     }
115     list.tail = list.tail.prev
```

```
116     list.tail.next = nil
117 }
118
119 type DoublyLinkedListNode struct {
120     key    string
121     value  int
122     prev   *DoublyLinkedListNode
123     next   *DoublyLinkedListNode
124 }
125
126 func (node *DoublyLinkedListNode) removeBindings() {
127     if node.prev != nil {
128         node.prev.next = node.next
129     }
130     if node.next != nil {
131         node.next.prev = node.prev
132     }
133     node.prev, node.next = nil, nil
134 }
135
```