

Solution 1

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

