Solution 1

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

using System.Collections.Generic;

public class Program {
  public class LRUCache {
    public Dictionary<string, DoublyLinkedListNode> cache = new Dictionary<string,
        DoublyLinkedListNode>();
    public int maxSize;
    public int currentSize = 0;
    public DoublyLinkedList listOfMostRecent = new DoublyLinkedList();

    public LRUCache(int maxSize) {
      this.maxSize = maxSize > 1 ? maxSize : 1;
    }

    // O(1) time | O(1) space
    public void InsertKeyValuePair(string key, int value) {
      if (!cache.ContainsKey(key)) {
        if (currentSize == maxSize) {
          evictLeastRecent();
        } else {
          currentSize++;
        }
        cache.Add(key, new DoublyLinkedListNode(key, value));
      } else {
        replaceKey(key, value);
      }
      updateMostRecent(cache[key]);
    }

    // O(1) time | O(1) space
    public LRUResult GetValueFromKey(string key) {
      if (!cache.ContainsKey(key)) {
        return new LRUResult(false, -1);
      }
      updateMostRecent(cache[key]);
      return new LRUResult(true, cache[key].value);
    }

    // O(1) time | O(1) space
    public string GetMostRecentKey() {
      return listOfMostRecent.head.key;
    }

    public void evictLeastRecent() {
      string keyToRemove = listOfMostRecent.tail.key;
      listOfMostRecent.removeTail();
      cache.Remove(keyToRemove);
    }

    public void updateMostRecent(DoublyLinkedListNode node) {
      listOfMostRecent.setHeadTo(node);
    }

    public void replaceKey(string key, int value) {
      if (!this.cache.ContainsKey(key)) {
        return;
      }
      cache[key].value = value;
    }
  }

  public class DoublyLinkedList {
    public DoublyLinkedListNode head = null;
    public DoublyLinkedListNode tail = null;

    public void setHeadTo(DoublyLinkedListNode node) {
      if (head == node) {
        return;
      } else if (head == null) {
        head = node;
        tail = node;
      } else if (head == tail) {
        tail.prev = node;
        head = node;
        head.next = tail;
      } else {
        if (tail == node) {
          removeTail();
        }
        node.removeBindings();
        head.prev = node;
        node.next = head;
        head = node;
      }
    }

    public void removeTail() {
      if (tail == null) {
        return;
      }
      if (tail == head) {
        head = null;
        tail = null;
        return;
      }
      tail = tail.prev;
      tail.next = null;
    }
  }

  public class DoublyLinkedListNode {
    public string key;
    public int value;
    public DoublyLinkedListNode prev = null;
    public DoublyLinkedListNode next = null;

    public DoublyLinkedListNode(string key, int value) {
      this.key = key;
      this.value = value;
    }

    public void removeBindings() {
      if (prev != null) {
```

```
        prev.next = next;
      }
      if (next != null) {
        next.prev = prev;
      }
      prev = null;
      next = null;
    }
  }

  public class LRUResult {
    public bool found;
    public int value;

    public LRUResult(bool found, int value) {
      this.found = found;
      this.value = value;
    }
  }
}

```