AlgoExpert Quad Layout C++ 12px Sublime Monokai 00:00:00

Prompt Scratchpad Our Solution(s) Video Explanation Run Code

Solution 1

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

#include <vector>
#include <unordered_map>
using namespace std;

class DoublyLinkedListNode {
public:
  string key;
  int value;
  DoublyLinkedListNode *prev;
  DoublyLinkedListNode *next;

  DoublyLinkedListNode(string key, int value) {
    this->key = key;
    this->value = value;
    this->prev = NULL;
    this->next = NULL;
  }

  void removeBindings() {
    if (this->prev != NULL) {
      this->prev->next = this->next;
    }
    if (this->next != NULL) {
      this->next->prev = this->prev;
    }
    this->prev = NULL;
    this->next = NULL;
  }
};

class DoublyLinkedList {
public:
  DoublyLinkedListNode *head;
  DoublyLinkedListNode *tail;

  DoublyLinkedList() {
    this->head = NULL;
    this->tail = NULL;
  }

  void setHeadTo(DoublyLinkedListNode *node) {
    if (this->head == node) {
      return;
    } else if (this->head == NULL) {
      this->head = node;
      this->tail = node;
    } else if (this->head == this->tail) {
      this->tail->prev = node;
      this->head = node;
      this->head->next = this->tail;
    } else {
      if (this->tail == node) {
        this->removeTail();
      }
      node->removeBindings();
      this->head->prev = node;
      node->next = this->head;
      this->head = node;
    }
  }

  void removeTail() {
    if (this->tail == NULL) {
      return;
    }
    if (this->tail == this->head) {
      this->head = NULL;
      this->tail = NULL;
      return;
    }
    this->tail = this->tail->prev;
    this->tail->next = NULL;
  }
};

class LRUCache {
public:
  unordered_map<string, DoublyLinkedListNode *> cache;
  int maxSize;
  int currentSize;
  DoublyLinkedList listOfMostRecent;

  LRUCache(int maxSize) {
    this->maxSize = maxSize > 1 ? maxSize : 1;
    this->currentSize = 0;
    this->listOfMostRecent = DoublyLinkedList();
  }

  // O(1) time | O(1) space
  void insertKeyValuePair(string key, int value) {
    if (this->cache.find(key) == this->cache.end()) {
      if (this->currentSize == this->maxSize) {
        this->evictLeastRecent();
      } else {
        this->currentSize++;
      }
      this->cache[key] = new DoublyLinkedListNode(key, value);
    } else {
      this->replaceKey(key, value);
    }
    this->updateMostRecent(this->cache[key]);
  }

  // O(1) time | O(1) space
  int *getValueFromKey(string key) {
    if (this->cache.find(key) == this->cache.end()) {
      return NULL;
    }
    this->updateMostRecent(this->cache[key]);
    return &this->cache[key]->value;
  }

  // O(1) time | O(1) space
  string getMostRecentKey() { return this->listOfMostRecent.head->key; }

  void evictLeastRecent() {
    string keyToRemove = this->listOfMostRecent.tail->key;
    this->listOfMostRecent.removeTail();
    this->cache.erase(keyToRemove);
  }

  void updateMostRecent(DoublyLinkedListNode *node) {
    this->listOfMostRecent.setHeadTo(node);
  }

  void replaceKey(string key, int value) {
    if (this->cache.find(key) == this->cache.end()) {
      return;
    }
    this->cache[key]->value = value;
  }
};
```