LeetCode LRU Cache: Hash + Doubly Linked List

Overview

The O(1) time algorithm to solve LeetCode LRU Cache is by combining hash table (access cache location) and doubly linked list (maintain the least recent).

LeetCode LRU Cache

Design and implement a data structure for Least Recently Used (LRU) cache. It should support the following operations: `get` and `set`.

`get(key)` - Get the value (will always be positive) of the key if the key exists in the

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

cache, otherwise return -1.
`set(key, value)` - Set or insert the value if the key is not already present. When the cache reached its capacity, it should invalidate the least recently used item before inserting a new item.

LeetCode LRU Cache Analysis: Hash + Doubly Linked List

Well, I personally do not consider this is a hard problem by utilizing the STL list from C++, or even we code all the things from scratch by writing the doubly linked list from scratch as long as we are careful for the specially cases like there is only one node, or we access the most recent node which does not require relocate any of the existed cache locations and so on. Anyways, here is how I come up with the final algorithm to implement the LRU cache.

1) Using a hash map to simulate the cache memory. This is very straightforward because we see key/value pairs, and how to maintain the most/least structure? My first try is to use an array/vector in C++ and if a key is hit (either get/set) then this key becomes the most recent, we will have to move/promote it to put it in the most recent location (either the head/tail in the array). If we use array, this promote operation inlcuding find and relocate costs O(N) time and it will report TLE time limit exceeds error

2) If we come up with the dirty version like 1), then one step further is very intuitive to think about what is the most efficient way to remove a node and put it into the tail / head of an

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

array like structure. Yes, it is linked list! And it should a doubly linked list with both head and tail information. Now all the operation would become O(1) time: for each (key, value), we make the doubly linked list node to store it and we store the address (pointer/memory address/iterator) of the cache location of the this pair (key, value). We can make either the head or the tail of the doubly linked list as the least recent cache location. And when some cache location is hit (either get or set), we need to relocate this node to the most recent location which could be done in O(1) time with doubly linked list. I have two versions of implementation, one is from scratch by implementing the whole doubly linke list and the other is to use STL list because at first I thought after erase the iterators are all invalidated so I do not trust STL list works exactly the same as what I expect, but it turns out the STL list is quite convenient, and I have another post “How C++ Iterator is Really Affected by erase(): vector vs list” talking about this issue for your reference. The following are the source codes both accepted by LeetCode OJ to pass this LRU Cache problem:

```
struct DListNode {
    int val;
    DListNode *next;
    DListNode *prev;
    DListNode(int x = 0) : val(x), next(NULL), prev(NULL) {}
};

class LRUCache{
   public:
```

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

```
LRUCache(int capacity = 1) : m_capacity(capacity), m_cache(), m_head(), m_ptrs() {}

int get(int key) {
    if (m_cache.find(key) == m_cache.end()) return -1;
    maintainLRU(key);
    return m_cache[key]; // safe
}
void set(int key, int value) {
    if (m_cache.find(key) == m_cache.end()) {
        if (m_cache.size() >= m_capacity) {
            int lru = m_head.next->val;
            auto itr = m_cache.find(lru);
            auto itr2 = m_ptrs.find(lru);
            m_cache.erase(itr);
            m_ptrs.erase(itr2);
            DListNode *curr = m_head.next;
            DListNode *prev = curr->prev;
            DListNode *next = curr->next;

            if (NULL == next) {
                delete curr;
                m_head.prev = m_head.next = NULL;
```

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

```
    }
    else {
        m_head.next = next;
        next->prev = &m_head;
        curr->prev = curr->next = NULL;
        delete curr;
    }
}
m_cache[key] = value;
DListNode *node = new DListNode(key);
if (NULL == m_head.prev) {
    m_head.next = node;
    node->prev = &m_head;
    m_head.prev = node;
}
else {
    m_head.prev->next = node;
    node->prev = m_head.prev;
    m_head.prev = node;
}

m_ptrs[key] = node;
```

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

```
            return ;
        }
        m_cache[key] = value;
        maintainLRU(key);
    }
private:
    void maintainLRU(int key) {
        DListNode *curr = m_ptrs[key];
        DListNode *prev = curr->prev;
        DListNode *next = curr->next;

        if (NULL == next) {
            return ;
        }
        prev->next = next;
        next->prev = prev;
        curr->prev = curr->next = NULL;
        m_head.prev->next = curr;
        curr->prev = m_head.prev;
        m_head.prev = curr;
    }
```

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

```
    private:
        unordered_map<int, int>        m_cache;
        DListNode                      m_head;
        unordered_map<int, DListNode*> m_ptrs;
        int                            m_capacity;
};
```

Well, if we are asked to code everything from scratch, the we have to do things like the above, and we could come up with more elegant code by using STL list as follows:

```
class LRUCache {
    public:
        typedef list<pair<int, int>>::iterator list_itr;
        LRUCache(int cap = 1) : capacity(cap), cache(), itrs() { }

        int get(int key) {
            if (itrs.find(key) == itrs.end()) return -1;
            promote(itrs[key]);
            return itrs[key]->second;
        }

        void set(int key, int value) {
            if (itrs.find(key) != itrs.end()) {
```

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

```
            itrs[key]->second = value;
            promote(itrs[key]);
            return ;
        }
        if (itrs.size() >= capacity) {
            auto itr = itrs.find(cache.back().first);
            cache.pop_back();
            itrs.erase(itr);
        }
        cache.push_front(make_pair(key, value));
        itrs[key] = cache.begin();
    }
private:
    void promote(list_itr loc) {
        cache.push_front(make_pair(loc->first, loc->second));
        itrs[loc->first] = cache.begin();
        cache.erase(loc);
    }
private:
    list<pair<int, int>> cache;
    unordered_map<int, list_itr> itrs;
    int capacity;
```

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

```
};
```

More discussion about the above codes:

1. The first one with everything from scratch costs 268 ms while the second STL list costs 340 ms, this is why C is faster than C++
2. Both get/set are counted as a hit and we need to relocate the accessed key/value to the most recent location.

Summary

The O(1) time algorithm to solve LeetCode LRU Cache is by combining hash table (access cache location) and doubly linked list (maintain the least recent).

Written on December 15, 2014

« How C++ Iterator is Really Affected by erase(): vector vs list

Git: .gitignore and quick way to stage many deleted files »

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com

Copyright © 2016 Sigmainfy. All Rights Reserved.

open in browser PRO version Are you a developer? Try out the HTML to PDF API pdfcrowd.com