146. LRU Cache (/problems/lru-cache/)

Average Rating: 3.83 (41 votes)

Feb. 26, 2019 | 65.8K views

Design and implement a data structure for Least Recently Used (LRU) cache (https://en.wikipedia.org/wiki/Cache_replacement_policies#LRU). It should support the following operations: `get` and `put`.

`get(key)` - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.
`put(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.

The cache is initialized with a **positive** capacity.

Follow up:
Could you do both operations in **O(1)** time complexity?

Example:

```
LRUCache cache = new LRUCache( 2 /* capacity */ );

cache.put(1, 1);
cache.put(2, 2);
cache.get(1);       // returns 1
cache.put(3, 3);    // evicts key 2
cache.get(2);       // returns -1 (not found)
cache.put(4, 4);    // evicts key 1
cache.get(1);       // returns -1 (not found)
cache.get(3);       // returns 3
cache.get(4);       // returns 4
```

Solution

Approach 1: Ordered dictionary

Intuition

We're asked to implement the structure (https://en.wikipedia.org/wiki/Cache_replacement_policies#LRU) which provides the following operations in $\mathcal{O}(1)$ time :

- Get the key / Check if the key exists
- Put the key
- Delete the first added key

The first two operations in $\mathcal{O}(1)$ time are provided by the standard hashmap, and the last one - by linked list.

> There is a structure called *ordered dictionary*, it combines behind both hashmap and linked list. In Python this structure is called *OrderedDict* (https://docs.python.org/3/library/collections.html#collections.OrderedDict) and in Java *LinkedHashMap* (https://docs.oracle.com/javase/8/docs/api/java/util/LinkedHashMap.html).

Let's use this structure here.

Implementation

Java Python Articles > 146. LRU Cache Copy

```python
from collections import OrderedDict
class LRUCache(OrderedDict):

    def __init__(self, capacity):
        """
        :type capacity: int
        """
        self.capacity = capacity

    def get(self, key):
        """
        :type key: int
        :rtype: int
        """
        if key not in self:
            return - 1

        self.move_to_end(key)
        return self[key]

    def put(self, key, value):
        """
        :type key: int
        :type value: int
        :rtype: void
        """
        if key in self:
```

Complexity Analysis

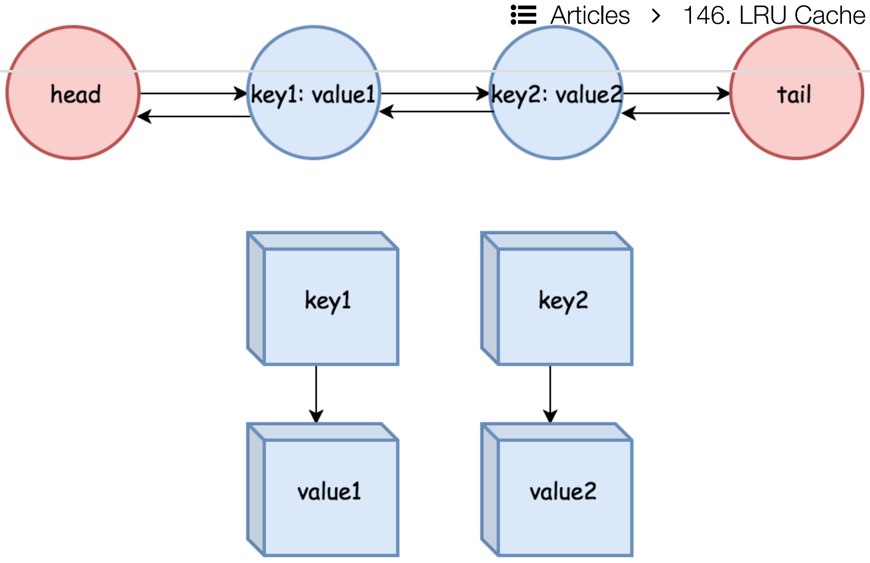
- Time complexity : $\mathcal{O}(1)$ both for `put` and `get` since all operations with ordered dictionary : `get/in/set/move_to_end/popitem` (`get/containsKey/put/remove`) are done in a constant time.
- Space complexity : $\mathcal{O}(capacity)$ since the space is used only for an ordered dictionary with at most `capacity + 1` elements.

Approach 2: Hashmap + DoubleLinkedList

Intuition

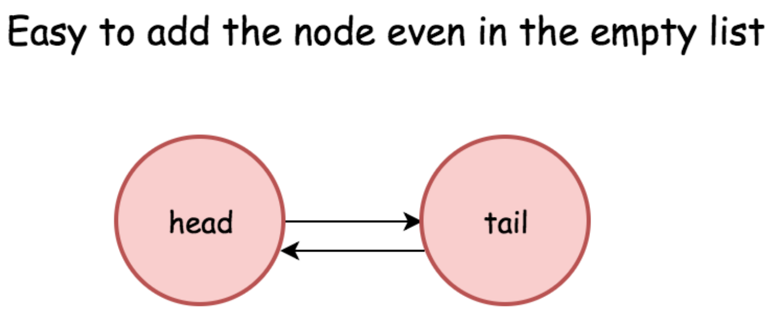
This Java solution is an extended version of the the article published on the Discuss forum (https://leetcode.com/problems/lru-cache/discuss/45911/Java-Hashtable-%2B-Double-linked-list-(with-a-touch-of-pseudo-nodes)).

The problem can be solved with a hashmap that keeps track of the keys and its values in the double linked list. That results in $\mathcal{O}(1)$ time for `put` and `get` operations and allows to remove the first added node in $\mathcal{O}(1)$ time as well.



One advantage of *double* linked list is that the node can remove itself without other reference. In addition, it takes constant time to add and remove nodes from the head or tail.

One particularity about the double linked list implemented here is that there are *pseudo head* and *pseudo tail* to mark the boundary, so that we don't need to check the `null` node during the update.



Implementation

Java Python

Articles > 146. LRU Cache

Copy

```
class DLinkedNode():
    def __init__(self):
        self.key = 0
        self.value = 0
        self.prev = None
        self.next = None

class LRUCache():
    def _add_node(self, node):
        """
        Always add the new node right after head.
        """
        node.prev = self.head
        node.next = self.head.next

        self.head.next.prev = node
        self.head.next = node

    def _remove_node(self, node):
        """
        Remove an existing node from the linked list.
        """
        prev = node.prev
        new = node.next

        prev.next = new
```

Complexity Analysis

- Time complexity : $\mathcal{O}(1)$ both for `put` and `get`.
- Space complexity : $\mathcal{O}(capacity)$ since the space is used only for a hashmap and double linked list with at most `capacity + 1` elements.

Analysis written by @liaison (https://leetcode.com/liaison/) and @andvary (https://leetcode.com/andvary/)

Rate this article:

Previous (/articles/longest-substring-with-at-most-k-distinct-characte/)

Next (/articles/count-univalue-subtrees/)

Comments: 21

Sort By

Type comment here... (Markdown is supported) Articles › 146. LRU Cache

Preview Post

SOL740 (sol740) ★ 94 February 26, 2019 1:50 PM
(/sol740)

Using an OrderedDict / LinkedHashMap to solve this problem in an interview setting is like using [::-1] / .reverse() to solve how to reverse a string, yeah you got it right but is that what the interviewer is looking for?

80 Share Reply

SHOW 5 REPLIES

asheth111 (asheth111) ★ 168 February 27, 2019 5:52 PM
(/asheth111)

I'd find a much better explanation and solution from the discussion section.

21 Share Reply

SHOW 1 REPLY

softwareshortcut (softwareshortcut) ★ 91 March 17, 2019 8:13 PM
(/softwareshortcut)

Java Solution. Not a huge fan of the code proposed for solution #2. Here is a more straight forward code (same approach, just cleaner code).

```
class Node {
  int value;
```

Read More

14 Share Reply

SHOW 4 REPLIES

yang-zhang-syd (yang-zhang-syd) ★ 7 March 12, 2019 3:25 AM
(/yang-zhang-syd)

why remove oldest? what if a cache is frequently used and the eldest the same time?

7 Share Reply

SHOW 4 REPLIES

liaison (liaison) STAFF ★ 4341 March 1, 2019 10:51 AM
(/liaison)

@KingXun (https://leetcode.com/kingxun) good question! The key in DLinkedNode would help us to remove the node itself from the cache at the moment the node becomes *stale* and is removed along with the invocation of the function `popTail`. We need to keep the key along with the node itself, because when the node is removed, we are not blessed with the key from the caller of LRU cache.

6 Share Reply

(/halfnibble)

halfnibble (halfnibble) ★ 7 March 12, 2019 1:12 AM

The description explicitly states, **Set or insert the value if the key is not already present.** And yet our solution must insert/update values when the key is already present. Am I the only person bothered by this apparent contradiction?

7 Share Reply

SHOW 3 REPLIES

(/sergiip)

sergiip (sergiip) ★ 8 March 18, 2019 10:06 PM

`Hashtable` in java? Seriously??! And then these people are getting offers instead of the real engineers and write that code in production.

7 Share Reply

SHOW 4 REPLIES

(/alexworden)

alexworden (alexworden) ★ 5 August 7, 2019 10:07 AM

I really like the dummy head & tail technique. Much more simple implementation with those in place and really minimal extra memory overhead. I just spent an hour catching all the edge cases without them! I'm puzzled why my solution is slower than most - I presume most solutions are cheating and using an LinkedHashMap. I also suspect the performance numbers are based on unrealistically small sample sets.

Read More

2 Share Reply

(/calvinchankf)

calvinchankf (calvinchankf) ★ 1143 June 25, 2019 6:07 PM

how come this question is `downgraded` as a medium question suddenly?

2 Share Reply

SHOW 1 REPLY

(/yh32)

yh32 (yh32) ★ 6 September 27, 2019 10:06 PM

Does the interviewer really expects you to write this much code in just 30 mins?

1 Share Reply

SHOW 1 REPLY

‹ 1 2 3 ›

Help Center (/support/) | Students (/students) | Terms (/terms/) | Privacy