146. LRU Cache ★

Question | Editorial Solution | My Submissions (/problems/lru-cache/submissions/)

Total Accepted: **89446** Total Submissions: **564352** Difficulty: **Hard**

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 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.

Subscribe (/subscribe/) to see which companies asked this question

Show Tags

Have you met this question in a real interview? Yes No

Discuss (https://leetcode.com/discuss/questions/oj/lru-cache)

Pick One (/problems/random-one-question/)

C++

```
class LRUCache {
    typedef pair<int, int> CacheNode;
public:
    LRUCache(int capacity):_capacity(capacity) {

    }

    int get(int key) {
        auto it = _hashTable.find(key);
        if (it == _hashTable.end()) return -1;

        auto cacheNode = *it->second;
        _cacheList.erase(it->second);
        _cacheList.push_back(cacheNode);
        _hashTable[key] = --_cacheList.end();
        return cacheNode.second;
    }

    void set(int key, int value) {
        auto it = _hashTable.find(key);
        if (it == _hashTable.end()) {
            if (_cacheList.size() == _capacity) {
                _hashTable.erase(_cacheList.begin()->first);
                _cacheList.erase(_cacheList.begin());
```

```
25              }
26              _cacheList.push_back(CacheNode(key, value));
27              _hashTable.insert(pair<int, list<CacheNode>::iterator>(key, --_cacheL
28          }
29          else {
30              it->second->second = value;
31              auto cacheNode = *it->second;
32              _cacheList.erase(it->second);
33              _cacheList.push_back(cacheNode);
34              _hashTable[key] = --_cacheList.end();
35          }
36      }
37  private:
38      list<CacheNode> _cacheList; // list<pair<key, value>>
39      unordered_map<int, list<CacheNode>::iterator> _hashTable; // key, listIter
40      int _capacity;
```

Notes

Submit Solution