146. LRU Cache <Medium>

class LRUCache {

private:

int capacity;

list<int> recent;

unordered\_map<int, int> cache;

unordered\_map<int, list<int>::iterator> pos;

void use(int key) {

if (pos.find(key) != pos.end()) {

recent.erase(pos[key]);

} else if (recent.size() >= capacity) {

int old = recent.back();

recent.pop\_back();

cache.erase(old);

pos.erase(old);

}

recent.push\_front(key);

pos[key] = recent.begin();

}

public:

LRUCache(int capacity): capacity(capacity) {}

int get(int key) {

if (cache.find(key) != cache.end()) {

use(key);

return cache[key];

}

return -1;

}

void set(int key, int value) {

use(key);

cache[key] = value;

}

};

class LRUCache {

unordered\_map<int, pair<list<int>::iterator, int>> dictionary; // for O(1) retrieval in `get`

list<int> cache; // for O(1) insertion or deletion in 'put'

int capacity;

public:

LRUCache(int capacity) {

this->capacity = capacity;

}

int get(int key) {

if (dictionary.find(key)==dictionary.end())

return -1;

// To push the retrieved element to first. (we used it recently, so this goes to more priority one)

cache.erase(dictionary[key].first);

cache.push\_front(key);

dictionary[key].first = cache.begin();

return dictionary[key].second;

}

void put(int key, int value) {

// not found in the cache

if (dictionary.find(key)==dictionary.end()){

// cache capacity is full, so least used element should be removed

if (cache.size() == capacity){

int last\_key = cache.back();

// remove last element from cache

cache.pop\_back();

// removing the last element from dictionary

dictionary.erase(last\_key);

}

}

else{

cache.erase(dictionary[key].first); // using the pointer to the element, it gets removed in O(1)

}

cache.push\_front(key);

dictionary[key] = {cache.begin(), value};

}

};