<https://takeuforward.org/data-structure/implement-lru-cache/>

code:

class LRUCache {

public:

class node {

public:

int key;

int val;

node \* next;

node \* prev;

node(int \_key, int \_val) {

key = \_key;

val = \_val;

}

};

node \* head = new node(-1, -1);

node \* tail = new node(-1, -1);

int cap;

unordered\_map < int, node \* > m;

LRUCache(int capacity) {

cap = capacity;

head -> next = tail;

tail -> prev = head;

}

void addnode(node \* newnode) {

node \* temp = head -> next;

newnode -> next = temp;

newnode -> prev = head;

head -> next = newnode;

temp -> prev = newnode;

}

void deletenode(node \* delnode) {

node \* delprev = delnode -> prev;

node \* delnext = delnode -> next;

delprev -> next = delnext;

delnext -> prev = delprev;

}

int get(int key\_) {

if (m.find(key\_) != m.end()) {

node \* resnode = m[key\_];

int res = resnode -> val;

m.erase(key\_);

deletenode(resnode);

addnode(resnode);

m[key\_] = head -> next;

return res;

}

return -1;

}

void put(int key\_, int value) {

if (m.find(key\_) != m.end()) {

node \* existingnode = m[key\_];

m.erase(key\_);

deletenode(existingnode);

}

if (m.size() == cap) {

m.erase(tail -> prev -> key);

deletenode(tail -> prev);

}

addnode(new node(key\_, value));

m[key\_] = head -> next;

}

};

Here get will take normally O(1) but O(n) can be as in collision.hence consider O(1)