146. LRU Cache

QuestionEditorial Solution
My Submissions

- Total Accepted: **85343**
- Total Submissions: **538203**
- 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.

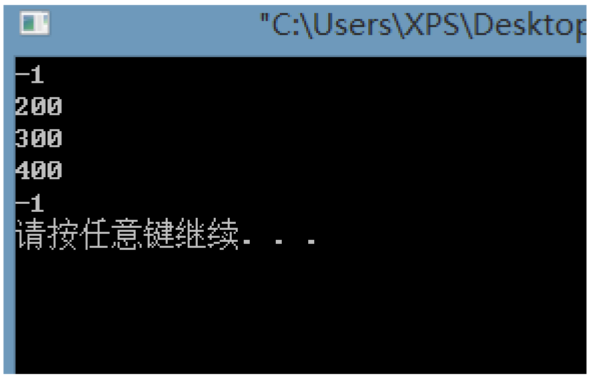
Subscribe to see which companies asked this question

```
//author:DemonMikalis
#include<sstream>
#include<iostream>
#include<algorithm>
#include<string>
#include<vector>
#include<map>
using namespace std;
struct node{
	struct node *prev;
	struct node *next;
	int key;
	int val;
	node(int key,int val)
	{
		this->key = key;
		this->val = val;
		prev = NULL;
		next = NULL;
	}
};
class LRUCache
{
	int capacity;
	typedef struct node *Node;
	map<int,Node> map1;
	int count;
	Node head,tail;
public:
	LRUCache::LRUCache(int capacity)
	{
		this->capacity= capacity;
		head = new node(0,0);
		tail = new node(0,0);
		head->next= tail;
		tail->prev= head;
		head->prev= NULL;
		tail->next= NULL;
		this->count=0;
	}
	LRUCache::~LRUCache()
	{
		Node tmp = head;
		while(tmp->next!=tail)
		{
			Node t = tmp;
			tmp = tmp->next;
			delete t;
		}
	}
	void LRUCache::deleteNote(Node node1)
	{
		node1->prev->next= node1->next;
		node1->next->prev= node1->prev;
	}
	void LRUCache::addToHead(Node node1)
	{
		node1->next = head->next;
		node1->next->prev= node1;
		node1->prev=head;
		head->next=node1;
	}
	int LRUCache::get(int key)
	{
		if(this->map1.find(key)!=map1.end())
		{
			Node nd = map1[key];
			int result = nd->val;
			this->deleteNote(nd);
			this->addToHead(nd);
			return result;
		}
		return -1;
	}
	void LRUCache::set(int key,int value)
	{
		if(map1.find(key)!=map1.end())
		{
			Node nd = map1[key];
			nd->val = value;
			this->deleteNote(nd);
			this->addToHead(nd);
		}else{
			Node nd = new node(key,value);
			map1.insert(make_pair<int,Node>(key,nd));
			if(count<capacity)
			{
				count++;
				this->addToHead(nd);
			}else{
				map1.erase(tail->prev->key);
				this->deleteNote(tail->prev);
				this->addToHead(nd);
			}
		}
	}
};
```

```
int main(int argc,char *argv[])
{
	LRUCache *lru = new LRUCache(3);

	lru->set(1,100);
	lru->set(2,200);
	lru->set(3,300);
	lru->set(4,400);

	int ans = lru->get(1);
	int ans2= lru->get(2);
	int ans3= lru->get(3);
	int ans4= lru->get(4);
	int ans5= lru->get(5);
	cout<<ans<<endl;
	cout<<ans2<<endl;
	cout<<ans3<<endl;
	cout<<ans4<<endl;
	cout<<ans5<<endl;
	delete lru;
	return 0;
}
```



"C:\Users\XPS\Desktop

```
-1
200
300
400
-1
请按任意键继续. . .
```