



Description

Solution

Discuss (999+)

Submissions

C#

146. LRU Cache

Medium

11614

458

Add to List

Share

Design a data structure that follows the constraints of a **Least Recently Used (LRU) cache**.

Implement the `LRUCache` class:

- `LRUCache(int capacity)` Initialize the LRU cache with **positive** size `capacity`.
- `int get(int key)` Return the value of the `key` if the key exists, otherwise return `-1`.
- `void put(int key, int value)` Update the value of the `key` if the `key` exists. Otherwise, add the `key-value` pair to the cache. If the number of keys exceeds the `capacity` from this operation, **evict** the least recently used key.

The functions `get` and `put` must each run in $O(1)$ average time complexity.

Example 1:

Input

```
["LRUCache", "put", "put", "get", "put", "get", "put", "get",  
"get", "get"]  
[[2], [1, 1], [2, 2], [1], [3, 3], [2], [4, 4], [1], [3], [4]]
```

Output

```
[null, null, null, 1, null, -1, null, -1, 3, 4]
```

Explanation

```
LRUCache lruCache = new LRUCache(2);  
lruCache.put(1, 1); // cache is {1=1}  
lruCache.put(2, 2); // cache is {1=1, 2=2}  
lruCache.get(1);    // return 1  
lruCache.put(3, 3); // LRU key was 2, evicts key 2, cache is {1=1, 3=3}  
lruCache.get(2);    // returns -1 (not found)  
lruCache.put(4, 4); // LRU key was 1, evicts key 1, cache is {4=4, 3=3}  
lruCache.get(1);    // return -1 (not found)  
lruCache.get(3);    // return 3
```

i

{}

↶

↷

```
1 using System.Collections.Specialized;  
2 public class LRUCache {  
3  
4     public class CacheEntry {  
5         public int Data;  
6         public DateTime IndexOnList {get; set;}  
7     }  
8     protected Dictionary<int, CacheEntry> dict = new Dictionary<int, CacheEntry>();  
9  
10    protected SortedDictionary<DateTime, CacheEntry> list = new SortedDictionary<DateTime, CacheEntry>();  
11    protected int CacheCapacity = 0;  
12  
13    public LRUCache(int capacity) {  
14        CacheCapacity = capacity;  
15    }  
16  
17    public int Get(int key) {  
18        if (dict.ContainsKey(key))  
19            return dict[key].Data;  
20    }  
21  
22    public void Put(int key, int value) {  
23        if (dict.ContainsKey(key)) {  
24            dict[key].Data = value;  
25            list.Remove(dict[key].IndexOnList);  
26            list.Add(DateTime.Now, dict[key]);  
27        } else {  
28            dict[key] = new CacheEntry { Data = value, IndexOnList = DateTime.Now };  
29            list.Add(DateTime.Now, dict[key]);  
30            if (list.Count > CacheCapacity) {  
31                list.Remove(list.First().Value.IndexOnList);  
32                dict.Remove(list.First().Value.Data);  
33            }  
34        }  
35    }  
36 }  
37
```

Your previous code was restored from y

Testcase

Run Code Result

Accepted

Runtime: 108 ms

Your input

```
["LRUCache", "put",  
[2], [1,1], [2,2],
```

Output

```
[null, null, nul
```

Expected

```
[null, null, null, :
```

Console

Use Example Testcase

Problems

Pick One

< Prev

0/0

Next >

Run Code ^

Submit