

LFU Cache

LRU

LRU

LFUcache (int capacity)

int get (int key)

void put (int key, int value)

get(1) (-1)

DS

Input

["LFUCache", "put", "put", "get", "put", "get", "get", "put", "get", "get", "get"]

[[2], [1, 1], [2, 2], [1], [3, 3], [2], [3], [4, 4], [1], [3], [4]]

Output

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

Explanation

// cnt(x) = the use counter for key x

// cache=[] will show the last used order for tiebreakers (leftmost element is most recent)

LFUCache lfu = new LFUCache(2);

lfu.put(1, 1); // cache=[1,], cnt(1)=1

lfu.put(2, 2); // cache=[2, 1], cnt(2)=1, cnt(1)=1

lfu.get(1); // return 1

// cache=[1, 2], cnt(2)=1, cnt(1)=2

lfu.put(3, 3); // 2 is the LRU key because cnt(2)=1 is the smallest, invalidate 2.

// cache=[3, 1], cnt(3)=1, cnt(1)=2

lfu.get(2); // return -1 (not found)

lfu.get(3); // return 3

// cache=[3, 1], cnt(3)=2, cnt(1)=2

lfu.put(4, 4); // Both 1 and 3 have the same cnt, but 1 is LRU, invalidate 1.

// cache=[4, 3], cnt(4)=1, cnt(3)=2

lfu.get(1); // return -1 (not found)

lfu.get(3); // return 3

// cache=[3, 4], cnt(4)=1, cnt(3)=3

lfu.get(4); // return 4

// cache=[4, 3], cnt(4)=2, cnt(3)=3



(2)

put (1,1)

put (2,2)

get (1)

put (3,3)

get (2)

get (3)

put (4,4)

get (1)

get (3)

get (4)

Input

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

Output

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

Explanation

```
// cnt(x) = the use counter for key x  
// cache=[] will show the last used order for tiebreakers (leftmost element is most recent)  
LFUCache lfu = new LFUCache(2);  
lfu.put(1, 1); // cache=[1,_], cnt(1)=1  
lfu.put(2, 2); // cache=[2,1], cnt(2)=1, cnt(1)=1  
lfu.get(1);    // return 1  
               // cache=[1,2], cnt(2)=1, cnt(1)=2  
lfu.put(3, 3); // 2 is the LFU key because cnt(2)=1 is the smallest, invalidate 2.  
               // cache=[3,1], cnt(3)=1, cnt(1)=2  
lfu.get(2);    // return -1 (not found)  
lfu.get(3);    // return 3  
               // cache=[3,1], cnt(3)=2, cnt(1)=2  
lfu.put(4, 4); // Both 1 and 3 have the same cnt, but 1 is LRU, invalidate 1.  
               // cache=[4,3], cnt(4)=1, cnt(3)=2  
lfu.get(1);    // return -1 (not found)  
lfu.get(3);    // return 3  
               // cache=[3,4], cnt(4)=1, cnt(3)=3  
lfu.get(4);    // return 4  
               // cache=[4,3], cnt(4)=2, cnt(3)=3
```

Solution Design

List<int> doubly linked

By using 3 Maps

add

keyVal: key, <value, freq>
↳ unordered_map <int, pair<int, int>> keyVal;

freqList: freq, key1 ↔ key2 ↔ key3
↳ unordered_map <int, list<int>> freqList;

pos: key, 0x10124xEO1
↳ Address of above node

↳ unordered_map <int, list<int>::iterator> pos;

[key ↔ address]

Two Variable

Capacity:
↳ To store capacity ✓

minfreq:
↳ to store minimum frequency. ✓

```
unordered_map<int, pair<int, int>>keyVal;  
unordered_map<int, list<int>> freqList;  
unordered_map<int, list<int>::iterator > pos;
```

(2)

$$\min \text{freq} = 1$$

Key

minfr →

$(3) \rightarrow (3)$
 $\times 6 \times 12$

Pos.

iterations

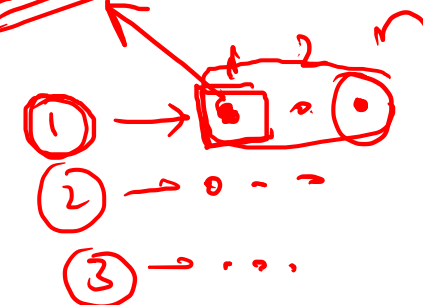
$$\frac{-(4)}{\times 1012}$$

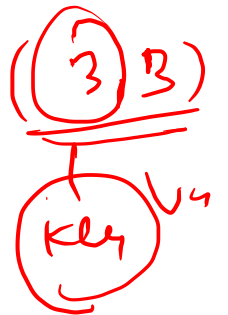
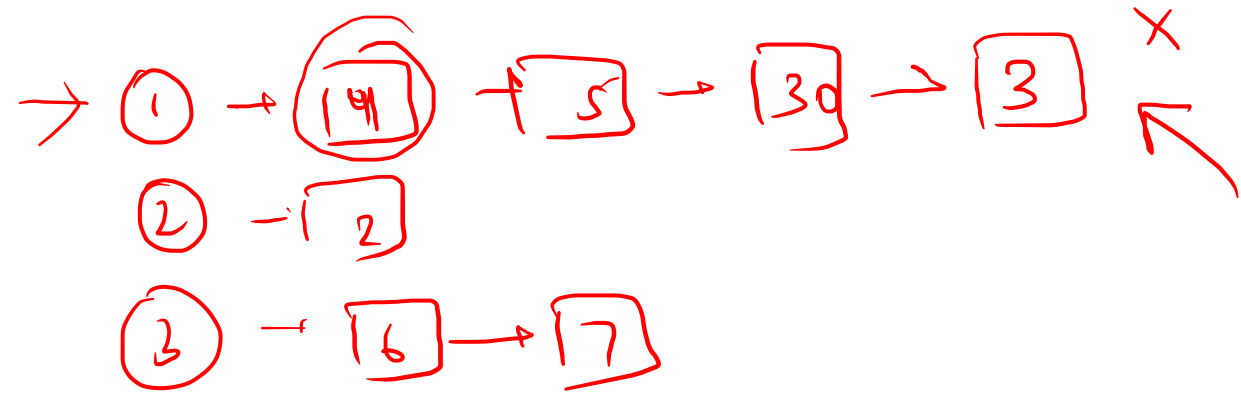
```

1. put (1, 1)
2. put (2, 2)
3. get (1)
4. put (3, 3)
5. get (2)
6. get (3)
7. put (4, 4)
8. get (1)
9. get (3)
10. get (4)

```

LRO





freq

① → ~~5~~ ✗

② → ③ → ⑥ → ⑤

get(5)

put(5, 9)

update

(increment frequency by 1)

we have key = 5

```
{ int curr_freq = KeyVal[key].second;  
  freqList[curr_freq].erase(pos[key]);
```

```
{ KeyVal[key].second++
```

```
{ curr_freq = KeyVal[key].second;  
  freqList[curr_freq].push_back(key);
```

```
{ pos[key] = --freqList[curr_freq].end();
```

```
if( freqList[min_freq].empty() )  
  min_freq++;
```

KeyVal : [1, (1, 1)]
 [5, (3, 1)]

min_freq : 1

freqList : → ① → 1 → 5 ✗
 XF014 XF015
 ↘
 ② → 5 ✗
 XG012

pos : (1) → XF014
 (5) → XF015

② → ⑥ → ③ → ⑤

if capacity
is full

int delkey = freqlist [minfreq]. front();

✓ keyval. erase (delkey)

delkey = 1

✓ pos. erase (delkey)

✓ freqlist [minfreq]. pop-front();

minfreq

freqlist:

① → ~~1~~ → 5
XF014 XF015

② → 3
XF016

③ → 6
XF041

Insert

keyval [key] = { value, 1 }

freqlist [1]. push-back (key);

pos [key] = -- freqlist [1]. end();

minfreq = 1