460. LFU Cache

Add to List
QuestionEditorial Solution
My Submissions

- Total Accepted: **3873**
- Total Submissions: **20387**
- Difficulty: **Hard**
- Contributors: **1337c0d3r** , **fishercoder**

Design and implement a data structure for Least Frequently Used (LFU) cache. It should support the following operations:get and put.

get(key) - Get the value (will always be positive) of the key if the key exists in the cache, otherwise return -1.

put(key, value) - Set or insert the value if the key is not already present. When the cache reaches its capacity, it should invalidate the least frequently used item before inserting a new item. For the purpose of this problem, when there is a tie (i.e., two or more keys that have the same frequency), the least **recently** used key would be evicted.

Follow up:

Could you do both operations in **O(1)** time complexity?

Example:

```
LFUCache cache = new LFUCache( 2 /* capacity */ );

cache.put(1, 1);
cache.put(2, 2);
cache.get(1);       // returns 1
cache.put(3, 3);    // evicts key 2
cache.get(2);       // returns -1 (not found)
cache.get(3);       // returns 3.
cache.put(4, 4);    // evicts key 1.
cache.get(1);       // returns -1 (not found)
cache.get(3);       // returns 3
cache.get(4);       // returns 4
```

```
class LFUCache {

public:

    int cap;

      int size;
```

```
   int minfreq;
   map<int,pair<int,int> > m;//key to {value,freq};
   map<int,list<int>::iterator > mIter; //key to list iterator
   map<int,list<int> > fm; //freq to key list
LFUCache(int capacity) {
    cap = capacity;
    size = 0;
}

int get(int key) {
    if(m.count(key)==0) return -1;
    fm[m[key].second].erase(mIter[key]);
    m[key].second++;
    fm[m[key].second].push_back(key);
    mIter[key] = --fm[m[key].second].end();
    if(fm[minfreq].size()==0) minfreq++;
    return m[key].first;
}

void put(int key, int value) {
    if(cap<=0) return;
   int storedValue = get(key);
   if(storedValue!=-1)
   {
          m[key].first= value;
          return;
       }
       if(size>=cap)
       {
          m.erase(fm[minfreq].front());
          mIter.erase(fm[minfreq].front());
          fm[minfreq].pop_front();
```

```
            size--;
        }

            m[key] = {value,1};
            fm[1].push_back(key);
            mIter[key] = --fm[1].end();
            minfreq=1;
            size++;
    }
};

/**
 * Your LFUCache object will be instantiated and called as such:
 * LFUCache obj = new LFUCache(capacity);
 * int param_1 = obj.get(key);
 * obj.put(key,value);
 */
```