# 460. LFU Cache

Hard 1362 125 Add to List Share

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.

Note that the number of times an item is used is the number of calls to the `get` and `put` functions for that item since it was inserted. This number is set to zero when the item is removed.

**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
```

```
1  class LFUCache
2
3  =begin
4      :type capacity: Inte
5  =end
6      def initialize(capa
7
8      end
9
10
11 =begin
12     :type key: Integer
13     :rtype: Integer
14 =end
15     def get(key)
16
17     end
18
19
20 =begin
21     :type key: Integer
22     :type value: Integer
23     :rtype: Void
24 =end
25     def put(key, value)
26
27     end
28
29
30 end
31
32 # Your LFUCache object w
   such:
33 # obj = LFUCache.new(cap
34 # param_1 = obj.get(key)
35 # obj.put(key, value)
```