LRU 完成思路

基本思路：采用封装的链表来进行排序。有因为在 list 内部，所有的检索必须使用 O（1）的循环来完成，所以采用将 list 内地址保存的方式，也就是空间换取时间的方式来减少复杂度。

伪代码：

链表结构体 「key, value, 前后的两个指针」

主体「

私有：容量，已有大小，链表的头尾指针，存地址的地图

公有：构造函数「创建头尾指针，清空地图，设置容量」

get 函数「判定链表中是否已经存在这个 key，

若存在，将之提取，放到队尾，同时返回其内部的 value

否则，返回 -1」

put 函数「判定是否存在

若存在，提取放到队尾，同时更新其内部的值

否则，创建新的链表节，放在队尾

同时，这一情况下，需判定是否已经超载，

如果超载，需要删去队首的链表节。

」

因链表节的先后放入顺序，直接导致链表头是最久未调用的

LFU 完成思路

基本思路： 1、采用封装结构体来贮存数值

2、同时构建 map 地图来实现依据于使用次数的分层。

3、因为同一复杂度情况下删除要求按照时间，此处使用链表

4、采用存储地址的方式来减少复杂度

伪代码：

结构体「key，value，frequent」

主体「

私有：容量，已有大小，最小频率，存指针的地图，存分层链表的地图，存链表地址的地图

公有：

构造函数「清空众 map，设置容量，给定初始大小和最小频率

get 函数「 判定是否已经存在对应的 key

若不存在，则返回-1

若存在，则将这一结构体的复杂度上调一级（直接地址）

更新时间位置，更新地址

同时如需要，删去原复杂度的部分

返回对应值

」

put 函数「 调用 get 函数

若返回具体数值，则说明已经存在，

同时 get 函数已经调整好了此节的对应位置

只需要改动其值就好了

否则，创建结构体

判定是否超量

如果超量，将最低复杂度层级的链表第一节拆下

更新最低复杂度为 1，

判定是否有复杂度为 1 的层级

如果有，尾端插入新建的结构体

否则，创建新的层级

更新地址