LRU:

数据结构：list(保存数据)，unordered\_map(查询数据)

单个节点：Key-Value, 访问次数，前后连接对象

成员变量：容量，头，尾，unordered\_map<Key, 节点指针>，智能指针(内存泄露)，

mutex全局锁

成员方法：initlist(创建头尾，构建链表)，

evictLeastRecent(删尾(removeNode)，map)，updateExisting(更新Value, moveToRecent)，moveToRecent(removeNode, inserNode)， addNew(判断容量(evictLeastRecent)，建新点，map, insertNode) insertNode(放头) removeNode(节点头尾)

类的初始化：initlist，类的析构默认，

Put: 判断容量<0, lock, find, updateExisting, addNew

Get: lock, find, moveToRecent,

Remove: lock, find, removeNode, map;

存在缓存污染的问题

LRU-k：继承LRU

成员变量：k(界限), LRU-k指针(保存未选用的Key的次数)，map(查询未选用的Key-Value)

成员函数：类的初始化：LRU的初始化，LRU-k指针的初始化，k

Get: LRU(查询)，在主缓存中，return; 不在主缓存中，LRU-k查询次数，次数增加1，次数大于k，map查找Key的Value, 如果能找到，放入主缓存中，LRU-k中删除，重新计数，map中删除，return Value;

Put: LRU(查询), 在主缓存中, LRU::put; 不在主缓存中，LRU-k查询次数，次数增加1，map中更新Value, 次数大于k，放入主缓存中Key-Value, LRU-k中删除，重新计数，map中删除

锁的力度比较大：

HashLRU：

成员变量：总容量， 切片数量，vector(LRU<Key, Value>指针)

成员函数：Hash函数

初始化函数: vector中构造切片数量的LRU；

Put：Key进行Hash映射，vector中对应放入Key-Value;

Get: Key进行Hash映射, LRU->get();

FreqList: 节点：Key, Value, nodeFreq, pre, next;

成员变量：listFreq, head, tail

成员函数：FreqList：初始化队列， freq；isEmpty：是否为空

addNode： 放头；removeNode：前后； getFirstNode：给出位于队尾的点

LFU： 成员变量：capacity\_， minFreq\_，maxAverageNum\_，curAverageNum\_，

curTotalNum\_， mutex\_，nodeMap\_， freqToFreqList\_

成员方法：decreaseFreqNum：减小curTotalNum\_， 更新curAverageNum\_；

kickOut：容量达到capacity\_时，找到minFreq对应FreqList的队尾元素，removeFromFreqList，nodeMap中删除Key-Value, decreaseFreqNum;

removeFromFreqList: 查询freq， freqToFreqList[freq]->removeNode;

addToFreqList: 查询node->nodeFreq, 查询freqToFreqList中是否有nodeFreq，如果没有重新创建，freqToFreqList[nodeFreq]->addNode

updateMinFreq: freqToFreqList中最小元素；

handleOverMaxAverageNum：从nodeMap中所有成员，removeFromFreqList，更新node->nodeFreq, 调用addToFreqList； updateMinFreq；

addFreqNum：更新curTotalNum\_， curAverageNum\_，判断是否调用handleOverMaxAverageNum；

getInternal：将node->Value付给引用Value, removeFromFreqList, nodeFreq++, addToFreqList, 判断是否是minFreq所在FreqList中的唯一元素， addFreqNum;

putInternal: 判断capacity是否已满，kickOut，nodeMap中添加， addToFreqList，addFreqNum，更新minFreq\_

类的初始化：capacity，minFreq，maxAverageNum，curAverageNum，curTotalNum；

Put：判断是否在nodeMap中，如果在则使用getInternal，否则使用putInternal；

Get: 判断是否在nodeMap中, getInternal;

ArcLru：

成员变量：capacity\_，ghostCapacity\_，transformThreshold\_，mutex\_，mainCache\_，ghostCache\_，

mainHead\_， mainTail\_，ghostHead\_， ghostTail\_

成员函数：initializeLists：初始化链表；addToFront：加入到主缓存的头部；

moveToFront：从链表中删除节点，addToFront；updateExistingNode：更新Value, moveToFront;

removeFromMain: 从主缓存中删除；removeFromGhost：从Ghost缓存中删除；

removeOldestGhost：链表，Cache删除最远节点；addToGhost：更新访问次数，加入到链表，Cache；

evictLeastRecent：从主缓存删除末尾，放入Ghost；addNewNode：构建新点到主缓存中；

updateNodeAccess：更新点，添加访问次数，判断是否达到transformThreshold；

Put：判断是否在主缓存中，是则updateExistingNode，否则addNewNode；

Get: 查询是否在主缓存中，获取访问次数，value；

checkGhost：查询是否在Ghost中，是则removeFromGhost，ghostCache中删除；

increaseCapacity：主缓存扩容；decreaseCapacity：主缓存缩容；

ArcLfu：

成员变量：capacity\_，ghostCapacity\_，transformThreshold\_，minFreq\_，mutex\_，

mainCache\_，ghostCache\_， freqMap\_， ghostHead\_，ghostTail\_；

成员函数：initializeLists：初始化Ghost链表；

updateNodeFrequency：节点访问次数+1，从原有链表中删除，放入链表；

updateExistingNode：更新Value, updateNodeFrequency；removeFromGhost：Ghost中删除节点；

removeOldestGhost：链表，ghostCache删除最后节点；addToGhost：放入Ghost链表头；

evictLeastFrequent：删除minFreq链表最后元素，加入到Ghost中；addNewNode：点入主缓存；

Put：判断是否在主缓存中，是则updateExistingNode，否则addNewNode；

Get：查询是否在主缓存中，获取Value；

checkGhost：查询是否在Ghost中，是则removeFromGhost，ghostCache中删除；

increaseCapacity：主缓存扩容；decreaseCapacity：主缓存缩容；

Arc：

成员变量：capacity\_， transformThreshold\_，lfuPart，lruPart；

成员函数：checkGhostCaches：判断Key位于谁的Ghost中，谁就扩容；