HW11 Doc

Name：陆昊成 StudentID：517021910649

LRU Cache

private member：
使用了一个名为 cap 的 int 作为对于 capacity 的记录。
使用了名为 record 的 list<pair<int,int> >作为对于 cache 的历史记录，越最近调用的越靠前。第一个 int 为 cache 的 key，第二个为 value。（增删的复杂度为 O(1)）
使用了名为 cache 的 unordered_map<int,list<pair<int,int> >::iterator>记录每个 cache 在 list 中的位置。第一个 int 为 cache 的 key。(使得 record 的查找的复杂度为 O(1)）

get：
在 get 时，实现了 O(1)的复杂度的算法。
查找 value，在 cache 中的复杂度为 O(1)。
修改历史记录，在 record 中将查找 cache 的位置放到 list 首位，复杂度为 O(1)。

put:
在 put 时，实现了 O(1)的复杂度的算法。
查找 input_cache 的 key 是否存在，若存在删除 record 中该 key 对应的 cache，复杂度为 O(1)。
在 record 的首位新增 put 进的<key,value>,在 cache 中增加<key,record::iterator>的索引，复杂度为 O(1)。

LFU Cache

private member：
使用了一个名为 cap 的 int 作为对于 capacity 的记录。
使用了一个名为 minFreq 的 int 作为对于频数最小值的记录。
使用了名为 cache 的 unordered_map<int, pair<int, int> >作为对于 cache 的历史记录，频数越高的越靠前，频数相同时，越最近调用的越靠前。第一个 int 为 cache 的 key，第二个为 value。（增删的复杂度为 O(1)）
使用了名为 freq 的 unordered_map<int, list<int> >作为对于 cache 的频数历史记录，频数相同的同一 list 中，越最近调用的越靠后。第一个 int 为 cache 的频数，第二个为该频数的 cache 的 key。（按频数查找和增删的复杂度为 O(1)）
使用了名为 freq_record 的 unordered_map<int, list<int>::iterator>记录每个 cache 在 freq 的 list 中的位置。第一个 int 为 cache 的 key。（使得 freq 中元素的查找的复杂度为 O(1)）

get：
在 get 时，实现了 O(1)的复杂度的算法。
查找 value，在 cache 中的复杂度为 O(1)。
修改历史记录，该 cache 的频数++，在 record 中将查找 cache 的位置放到下一频数的末尾

位，如果当前最低频数不存在，则最低频数++，复杂度为O(1)。

put：

在put时，实现了O(1)的复杂度的算法。

如果capacity>0，查找cache中的key是否存在。

若存在，则改变cache中该key对应的value，复杂度为O(1)。

若cache大小超过capacity，则在freq中频数为1的区域的首位删除一个cache，新增put进的<key,value>，记录频数为1，在cache中增加<key,record::iterator>的索引，并将minFreq重置为1，复杂度为O(1)。