Homework #3 (10.29)

1、假设我们采用 LRU 作为缓冲区置换策略，当我们向 Buffer Manager 发出一个读页请求时，请讨论一下：

⑴ 如果页不在缓冲区中，我们需要从磁盘中读入该页。请问如何才能在缓冲区不满的时候快速地返回一个 free 的 frame？请给出至少两种策略，并分析一下各自的时间复杂度。

⑵ 如何才能快速地判断所请求的页是否在缓冲区中？如果请求的页在缓冲区中，如何快速返回该页对应的 frame 地址？请给出至少两种策略，并分析一下各自的时间复杂度。

2、LRU 算法只考虑了最近一次访问时间，但在实际 DBMS 中，如果被置换的 frame 是 dirty 页，Buffer Manager 需要将该 frame 写回磁盘，这会带来额外的 I/O。如果被置换的 frame 是 clean 页，则无需将其写回磁盘，就不会产生额外 I/O。如果我们希望在选择置换页不仅考虑最近一次访问时间，而且还考虑 frame 的 dirty 属性，从而得到一种优化的写友好的LRU 算法，即优先置换 clean 的 frame。请试着设计一下基于上述思路的写友好的 LRU 策略：

⑴ 请给出修改后的 LRU 链表结构（是否需要增加新的指针或者数据结构）

⑵ 简要解释写友好的 LRU 置换页选择过程

⑶ 这种写友好的 LRU 策略在实际应用中的性能是否一定优于传统 LRU？为什么？

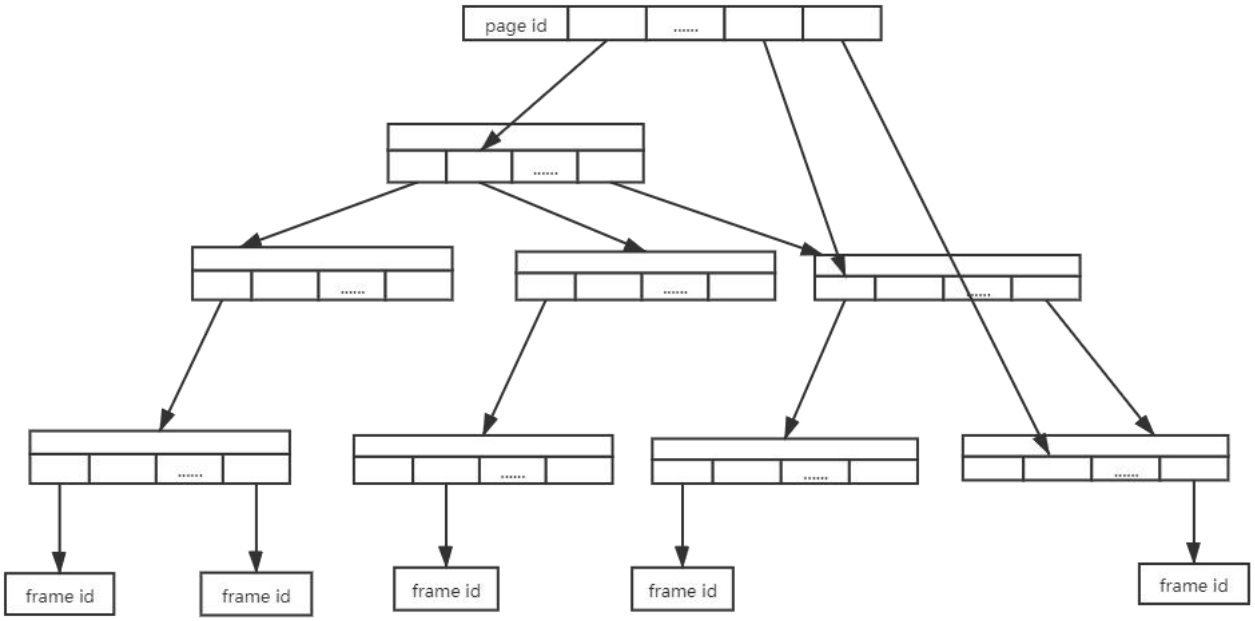
解：1.

（1）策略1：遍历缓冲区，寻找到一个空闲的 frame，然后返回，这种方法的时间复杂度是O(n)。

策略2：用两个链表将空闲的frame和已使用的frame在逻辑上分开存储为空闲链表和已使用链表，每次需要一个空闲的frame时，只需访问空闲链表找到一个空闲frame返回即可，时间复杂度为O(1)，但是每次返回空闲frame时，都要将新写入数据的frame从空闲链表指针指向已使用链表，需要额外的维护链表操作。

（2）

策略1：使用查找算法，为了高效地在缓冲区中查找页面，这里采用树结构——基数树（Radix Tree）这一数据结构到Page-frame管理中来。Radix tree能够根据文件索引结点和文件内逻辑偏移快速地定位page在缓冲区的位置，将page id转化为指向frame id的指针(具体来说是转化为一些列有指针组成的路径)，这是通过把id分段后作为各层节点的指针数组（以下将指针数组的项称为slot）的索引而达到检索的目的。分段通常使用将id右移指定位数后和指定长度的位掩码进行&运算获得，比如一个32位的id值，按4位一分段的方法，可以化为8个位串(每个含4位)，从高位到低位分别作为1~8层节点的slot索引，通过上一层节点的slot索引得到指向下一层节点的指针，如此直到最后一层，此时索引指向frame id，查找的时间复杂度接近O(logn)。逻辑结构如下：



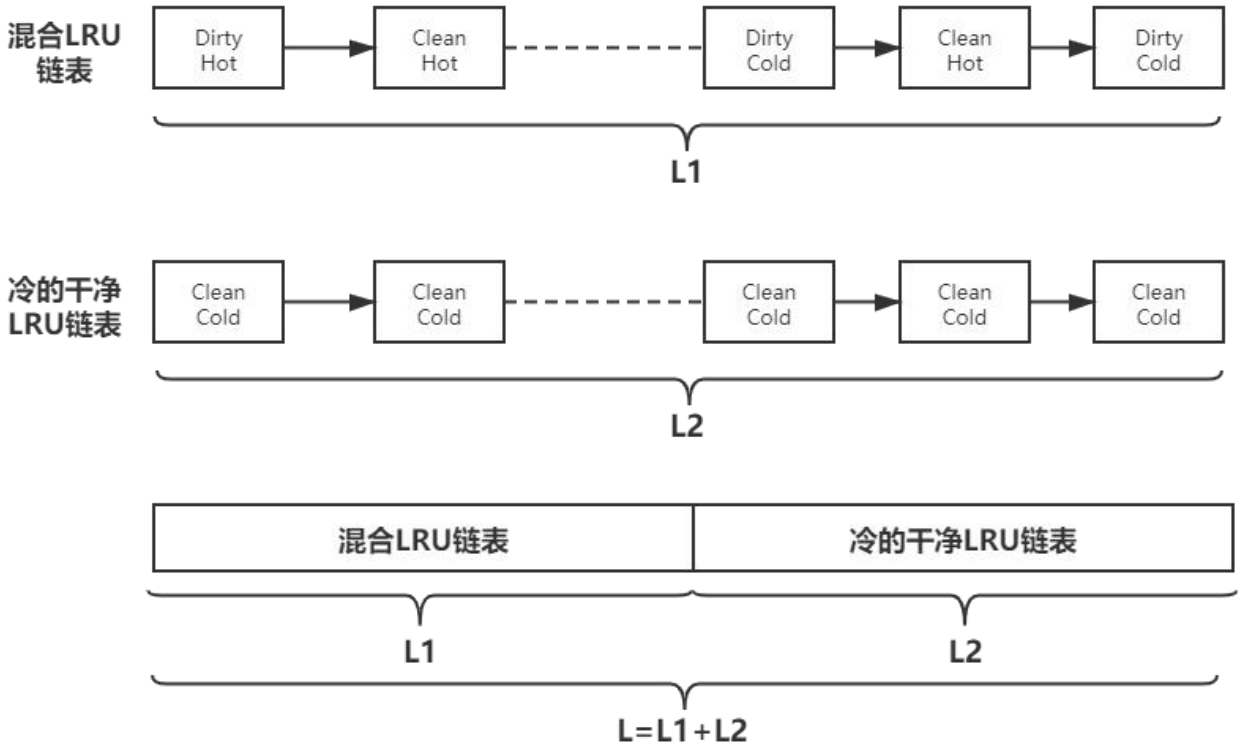
策略2：使用哈希算法，为了便于快速的找到某一个block或frame，缓冲区里面的block被组织到一些哈希表，哈希表中的指针代表一个链表，该链表所包含的所有节点均具有相同的哈希值，在该链表中查找可访问到指定的数据。比如建立一个BCB hTable[BufferSize]进行page到frame的映射，hash函数为H(k)=(page_id)%(buffersize)，通过请求页的page id通过哈希表快速映射到缓冲区的frame id，如果该页不在缓冲区中，则在哈希表中找不到该页对应的的frame id，查找时间复杂度接近O(1)。

2.

（1）**写友好的 LRU 策略**：使用双LRU链表维护缓冲区中的数据，其中，一个混合LRU链表维护缓冲区中的dirty页和热的clean页，而另一个冷的干净LRU链表只维护缓冲区中的冷的clean页。当缓冲区空间不足时，优先置换出冷的干净LRU链表中最近最少访问的clean页，否则当冷的干净链表为空时置换出混合的LRU链表中冷的dirty页。利用这一策略，可以有效地解决缓冲区一次性扫描式访问污染问题，减少了DBMS的写操作，同时还保持了较高的命中率，从而提高了DBMS的访问性能。

需要增加新的数据结构，将缓冲区中的frame依据其访问特点划分为冷的clean页、冷的dirty页、热的clean页和热的dirty页。在这种冷数据检测方法中，缓冲区中的每个frame都被赋予一个冷数据标记位：cold flag，当cold flag被置为1的时候，表明该frame为冷数据页，当cold flag被置为0时，表明该frame为热数据页。此外，当缓冲区中的冷数据页被再次访问时，要将其标记为热数据，即将cold flag置为0。

链表结构如下图所示，新的LRU算法维护了双缓冲区LRU链表，分别为混合LRU链表和冷的干净LRU链表，其中，混合的LRU链表维护缓冲区中的脏数据页和热的干净数据页，长度为L1，而冷的干净LRU链表只维护缓冲区中的冷的干净数据页，长度为L2。此外，如果缓冲区只能容纳L个数据页，则L1和L2的取值区间为[0, L]并且两者之和等于L。



（2）在新LRU算法中，默认初次被加载到缓冲区中的frame为冷数据页并将其放入冷的干净页链表的首部。当冷的干净LRU链表中的frame被再次访问或者变为脏数据页时，我们需要将其从冷的干净LRU链表中移到混合LRU链表的首部。当混合的LRU链表中的frame被再次访问时，我们需要将其移到混合LRU链表的首部。当缓冲区已满并且不存在满足访问条件的数据页时，我们需要顺序执行以下置换策略：

①查看冷的干净LRU链表是否为空，若不为空，直接将链表尾部的frame置换出去，否则执行步骤②。

②选择混合LRU链表尾部的frame作为候选替换frame，若该候选替换frame为冷的dirty页，则直接将其置换出去；若该候选替换frame为热的dirty页，首先将其移到混合LRU链表的首部并将cold flag置为1，然后选择混合LRU链表尾部的frame作为新的候选替换frame；若该候选替换frame为热的clean页，首先将其从混合LRU链表移到冷的干净LRU链表的首部并将cold flag置为1，然后选择混合LRU链表尾部的frame作为新的候选替换frame。若遍历完混合LRU链表以后，仍没有找到合适的替换frame，则需要再次执行置换策略。

（3）首先，这种新的LRU算法在的**大部分情况下**的效率是大大优于传统LRU算法的，由于最近一段时间内只访问一次的clean页初始将会被加载到冷的干净LRU链表并且新的LRU算法优先置换出冷的干净LRU链表中的frame，所以可以有效地防止最近一段时间内只访问一次的clean页污染混合的LRU链表。此外，当发生置换且缓冲区中不存在冷的clean页时，新的LRU算法并不立即置换出缓冲区中的热的clean页而是再给它们一次停留在缓冲区的机会，如果该frame很快将被访问（因为它很热，所以很快被访问的可能性会很大），将会节省一次读操作代价，否则该数据页将会被很快置换出去，不会带来任何额外的代价。

但是新的LRU算法仍存在一些问题，并不是在所有情况之下都优于传统LRU算法，当新的LRU算法中的冷的clean页链表为空时，会导致刚被加载到缓冲区中的clean页很快就会被置换出去，从而使得冷的干净LRU链表的长度在0和1之间跳跃，此时新的LRU算法将无法很好地执行冷的clean页优先置换策略。当缓冲区空间不足时，新的LRU算法很可能置换出刚被加载缓冲区中并且即将被频繁访问的clean页，而不管缓冲区中是否还存在许多冷的dirty页，这会使得缓冲区的命中率急剧下降，进一步影响了存储器的整体访问性能。当上层应用的写操作局部访问性高于读操作局部访问性时，这种影响尤为明显。