Lab 2

牛梓雄 521120910251

1. 代码逻辑

在 Lab 1 中，我们并未考虑 BufferPool 的最大页数限制。因此，Exercise 1 要求我们实现一种驱逐策略，当 BufferPool 满了以后，将其某一页进行替换处理。我采用 LRU（less recently used）策略，因为其实现起来较为方便。

- 在 Lab1 的 Map_ID_Page 的基础上，再建立一个由 ID 到整数的映射 LRUmap，如果该整数越大，则表示访问的时间越近，我们每次替换的是对应整数越小的 Page。
- LRUupdate(PageId pid)函数，更新给定页面的最近使用时间，实施 LRU 缓存替换策略。遍历所有页，减少它们的使用计数（除了当前访问的页），并将当前访问的页的计数设置为最高（numPages）。
- getPage(TransactionId tid, PageId pid, Permissions perm)也进行了相应修改。首先检查页是否在缓冲池中，如果不在且缓冲池已满，则逐出一页，然后加载新页。同时更新 LRU 信息。
- insertTuple(TransactionId tid, int tableId, Tuple t)实现了元组的插入，先用 Dbfile.insertTuple()函数判断哪些页被该改变了，标记其为脏页，并更新 LRUmap。deleteTuple 函数也采用类似操作。
- evictPage 函数遍历 LRU 映射以找到最少使用的页面。从缓冲池中逐出最少使用的页面。flushPage 和 flushAllPages 函数实现将脏页数据写回磁盘。flushPage 方法将指定的脏页写回其对应的数据文件。flushAllPages 方法遍历所有缓冲池中的页面，并调用 flushPage 方法写回所有脏页。discardPage 函数首先尝试将页面写回磁盘，然后从 Map_ID_Page 和 LRUmap 中移除其映射。

Exercise 2 要求我们实现了 B+数的 findLeafPage()函数以及 IndexPredicate.java 和 Predicate.java 以支持比较操作，通过测试（这两个 java 实现偏简单不赘述）。

- findLeafPage 函数实现了在 B+树中查找含有特定字段值 f 的叶子页面。

——首先检查传入的页面 ID pid 是否为叶节点类型；如果是，直接通过 getPage 方法返回该页。若不是叶节点，则假设为内部节点，使用 READ_ONLY 权限调用 getPage 获取该内部节点，并使用迭代器遍历其条目。

——如果 f 为 null，则选择第一个条目的左子节点继续搜索。如果 f 非空，函数将比较 f 和条目键的大小来确定继续搜索的方向：如果 f 大于当前条目键并且还有其他条目，继续向右遍历；如果 f 小于等于当前条目键，或者是最后一个条目，则根据比较结果选择左子节点或右子节点。通过这种方式，函数递归地向下搜索直到找到包含字段 f 的叶节点。

Exercise 3 要求我们实现 BTreeFile.splitLeafPage()和 BTreeFile.splitInternalPage()函数。这两个方法用于在 B+树中进行页面的拆分操作。

- splitLeafPage（） 函数主要用于在 B+ 树中分裂一个叶子节点。当一个叶子页面因插入操作而变得过满时，此函数将该页面分裂为两个页面，并将中间的键值向上传递到父节点中，如有必要，父节点也可能被递归地分裂。

——首先通过调用 getEmptyPage 方法获取一个新的空白叶子页面。然后使用反向迭代器遍历当前叶子页面的元组，并将大约一半的元组移动到新创建的叶子页面中。

——接着设置新叶子页面的左兄弟为当前页面，右兄弟为当前页面的原右兄弟。更新当前页面的右兄弟为新页面。如果当前页面有原右兄弟，更新原右兄弟的左

兄弟为新页面。

——选择一个合适的键（一般为移动到新页面的第一个元组的键）作为分界键，将其插入到父节点中。如果父节点没有足够空间，递归调用父节点的分裂操作。更新父节点的子页面指针，确保它们指向正确的子页面。

——最后根据给定的键 field，判断元组应插入原叶子页面还是新叶子页面，并返回相应的页面。

- splitInternalPage()函数实现将溢出的内部节点分裂成两个节点，并将分裂出的中间键“推上”到父节点中。

  ——首先通过 getEmptyPage ()获取一个新的内部节点页面，然后使用反向迭代器从原节点中移动一半的条目到新节点中，选取中间键作为上升到父节点的分界键。

  ——函数删除中间键的右子指针并在父节点中插入一个新的条目以链接新的和旧的内部节点

  ——根据插入的键值决定返回新节点或原节点。

Excercise4、5 要求实现关于“steal”和“merge”的函数，以实现 B+树的平衡。

- stealFromLeafPage()函数实现在 B+ 树的两个相邻内部节点之间重新分配元组以平衡它们的填充度。

  ——使用迭代器根据是从右兄弟还是左兄弟节点移动元组来遍历并收集要移动的元组，然后从源节点删除这些元组并将它们插入到当前节点中。

  ——完成元组移动后，更新父节点中的相应条目以反映节点中的最小或最大键值的变化。

- stealFromLeftInternalPage()函数实现从 B+ 树中一个节点的左兄弟节点窃取条目以重新平衡树。

  ——通过迭代器从左至右遍历左兄弟的条目，选取需要移动的条目，并更新与这些条目相关的父节点条目的键值。

  ——将父节点的键降至当前页面，并将窃取的条目（除最后一个外）插入当前页面。

  ——更新所有受影响节点的父指针，确保整个树的结构正确性和一致性。

  stealFromRightInternalPage()类似则不赘述。

- mergeLeafPages() 实现叶节点的合并操作。该函数将右叶节点中的所有元组移动到左叶节点，并更新相关的兄弟指针，使得右叶节点的右兄弟成为左叶节点的新右兄弟，进一步调整这些节点的兄弟链接以维持链表的连续性。接着，释放右叶节点以供后续重用，并在父节点中删除代表这两个正在合并的页面的条目。
- mergeInternalPages() 函数实现 B+ 树中合并两个内部节点。

  ——首先将所有从右节点的条目移动到左节点，并在移动过程中更新被移动条目的子节点的父指针，确保这些子节点指向正确的父节点。

  ——将父节点中的分割键（即原本用来分隔这两个节点的键）下移到左节点，填补两个节点合并后在左节点中产生的新空隙。

  ——右节点被清空并标记为可重用，同时从父节点中删除代表这两个正在合并的节点的条目。

2. 实验总结

本次实验共花费了我 5 天时间。本次实验的码量和难度都远大于 Lab1，理解 B+树的分裂合并插入等操作花费了本人较多的时间和经历。