PA2 Write-up

Team members: Wei-Hsiang Lin and Alex Wong

1. Describe any design decisions you made. These may be minimal for pa2.

1.1 Eviction Policy

We tried several eviction policies, for example, randomly evict, LRU (Least Recently Used) and MRU (Most Recently Used). In the end, we decided to keep LRU and leave MRU's implementation in comments (in the evictPage method).

In terms of design, we used a Doubly LinkedList (of PageIds) to keep track of page recency. The head of the LinkedList always stores the most recent PageId, and the tail stores the least recent. For convenience, we created a private method called 'updateRecency' to add the requested PageId to the head (if it does not exist in _pageRecencyList) or update the requested pageId's position to the head of LinkedList (if it does exist in _pageRecencyList). This method will be useful when we call getPage(...). When getPage(..., PageId pid, ...) is called, if pid(the page id) exists in the _pagePool, then we update pid in the Doubly LinkedList to the head position via updateRecency(..) (as explained earlier). However, if pid does not exist, then we need to check if _pagePool is full and evict page if it is. Then, we need to get the associated heap page, update it in the pagePool and update the Doubly LinkedListed via updateRecency(...) (as explained earlier).

1.2 Search in B+ Tree

In findLeafPage(...), the base case is finding a leaf page and returning it. However, if we land on an internal page, then we use an iterator to iterate the entries on the page, then we compare the keys using the Predicate class. When we find the appropriate child page, we recursively call the child page via findLeafPage(..) to eventually reach the leaf page. In this function, we identify the page type or category via BTreePageId's pgcateg() method and we get the BTreeLeafPage via the getPage(..) method.

1.3 Insert in B+ Tree

There are two types of insert; splitLeafPage and splitInternalPage. In splitInternalPage, we first create an empty sibling page via getEmptyPage(..) method. Then, we find the middle entry by using a regular and a reverse iterator that start on both ends and when they reach the same or a visited entry (depending on odd or even length), we know we have reached the middle entry. Next, we copy the right-half of the page to a newly created page. We iterate the original page from the entry after the middle entry, and at the same time, delete it from the original page and add it to the new page. Then, we update the parent page with the middle entry and parent pointers. Finally, we return the page that which an entry with key field "field" should be inserted based on the comparison results with the middle entry. In splitLeafPage, the routine is very similar, but we copy the middle entry to the new page, in addition to moving it to the parent page.

1.4 Delete in B+ Tree (Extra)

1.4.1 Steal Operations

There are two types of stealing operations; stealFromLeafPage and steal from internal pages (via stealFromLeftInternalPage or stealFromRightInternalPage method). I will start with stealFromLeafPage. In order to keep the tree balanced, we first need to know how many nodes to steal when we steal from sibling pages. This is calculated by (siblingAmt - currPageAmt)/2. Next, depending on whether we are stealing from leaf or right sibling, our sibling iterator could be a regular or reverse iterator. Next, we iterate the sibling iterator, deleting tuples from sibling and inserting into our current page in a for-loop. Lastly, we update parent entry depending on whether we are stealing from leaf or right sibling.

Stealing from internal pages work similarly, however, instead of updating the parent entry via a copy method, we are replacing/rotating it. We also need to update the child's parent page pointer as well.

1.4.2 Merge Operations

There two types of merge operations; mergeLeafPages and mergeInternalPages. For merge operations, it's simply just moving the right page entries to left page, but we need to make sure to change the parent pointer of the moved entries' children accordingly. Once the we finish moving the entry, we need to delete the empty page, and finally updating the parent entry.

2. Discuss and justify any changes you made to the API.

None

3. Describe any missing or incomplete elements of your code.

We copied the tests from "test extra" into our test folder, and completed all parts required to perform both ant test and ant systemtest. Due to time limitation, we managed to pass all the test, except for `testReuseDeletePages`.

4. Describe how long you spent on the assignment, and whether there was anything you found particularly difficult or confusing.

- As a team, we spent around 4 days to complete.
- The most challenging task was getting lost in the details. We were able to figure it out after we drew it out visually.

Notes before you submit (DELETE THIS BEFORE SUBMISSION)

3.2. Submitting your assignment

To submit your code, please create a zip file name it as follows: **cs660fall20-firstname-lastname-pa1.zip** and submit it through gradescope. Place the write-up in a file called **answers.txt or answers.pdf** in the top level of your CS660-pa1 directory. If you worked in a group, make a **GROUP submission**. Also DO NOT FORGET to mention all the name of contributors.