Project 1 Report

Overall Status

Current status: Completed

We implemented the major components in a way that followes the described algorithms in class. Our traversal algorithm is the same as the one shown in class (left child tree if < and right child tree if >=)

The Insert() function first check whether the root is null, then passes operation to _insert(). _insert() is a recursive function which runs for each index/leaf page. If the current page is an index page, it will search the values within the page and it will run _insert() on the coresponding child-page. If the current page is a leaf page, the value is inserted if possible. If inserting is not possible, it splits the leaf page by making a new leaf page and splitting the values between the two. A KeyDataEntry with a copied value is then pushed up the calling _insert() function. If the _insert() function is running on an insert page and it recieves a value from its sub-_insert() call, it inserts it into the current page. If no space exists it splits the index page similar to an leaf page but pushes the middle value instead of copying it. Lastly if the _insert() call on the root returns a value, a new root needs to be made with records pointing to the previous page and the newly created page from the sub-_insert() call.

The NaiveDelete() function works by first searching the tree similar to the _insert() function but with a while(type==index) loop instead of being recursive. When at a leaf page the leafPage.delEntry() function is called and its return value is used to determine whether the value exists. If the value exists, a while loop is executed to delete all values with the same key value.

Implemented Correctly

Insert(), _insert(), and NaiveDelete() are believed to be working properly

Unfinished Work

None.

File Descriptions

No new files were created. Only changes are in the designated area in BTreeFile.java.

Division of Labor

Initially we divided the labor by getting together and completing the required functions in the order which they are needed. This is Insert(), _insert(), then NaiveDelete(). This work was done in a pair programming fasion so many logical errors could be avoided on the first pass.

After the second coding session, Jacob decided to complete the rest of _insert() and NaiveDelete(). This code was then tested by both members to confirm that it works like it should.

- Landon Moon
    - 2 hours - 2/1/24 - Reading docs and started on Insert()

    - 2 hours - 2/8/24 - Finished Insert(). Helped polish and bugfix _insert()
    - 1 hour - 2/11/24 - Report writing
- Jacob Holz
    - 2 hours - 2/1/24 - Reading docs and started on _insert()
    - 2 hours - 2/8/24 - Completed rough draft of _insert() index page traversal/splitting
    - 2 hours - 2/9/24 - Completed rough draft of _insert() leaf insertion/splitting
    - 2 hours - 2/10/24 - Completed _insert and rough draft of NaiveDelete()
    - 2 hours - 2/11/24 - Complete NaiveDelete and test all methods
    - 1 hour - 2/12/24 - Report writing

Logical errors and how you handled them

Some logical errors were avoided before they cropped up through the use of pair programming, and reviewing line by line.

- Logical Error 1: index page splitting

  After looking through the provided library, we found out that there was no getCurent(rid) function in BTIndexPage. Both the getFirst(rid) and getNext(rid) are intended to be used as iterators but because we are changing the underlying data, this would be ill-advised. When splitting an index page, records from the first page needs to be removed and inserted into the second page. Without in-depth knowlege of the functions above, an iterator would give unusual results when deleting parts of the original array. This logical error was avoided by using getNext(rid) as an accessor by setting rid to the previous slotNo. This gave us more control over which rid we were copying and deleting

  There is probably a way to use getRecord(rid) instead of our implementation, but converting a custom Tuple class to a KeyDataEntry class felt overboard.

- Logical Error 2: Off by 1 error in above implementation

  When using getNext in this way, we overcompensated for the fact that it gets the next value by one each time it was used to get the middle value. This resulted in one entry being left over when entries were moved over resulting in invalid B+ trees upon testing.

  Solved easily by increasing the preMiddleSlotNumber by 1 and renaming it middleSlotNumber. This resulted in moving the window to be moved 1 index over which is correct.

- Logical Error 3: NaiveDelete only deleting duplicates in the same page

  When there were multiple pages containing the same value, only the last page would have the entries deleted. This was fixed by interating back through the pages until there stopped being matching entries. This case only occures when there are many many duplicates of the exact same entry.