B+ Tree Implementation

CSE 5331 DBMS Models and implementation

Instructor: Sharma Chakravarthy
GTA: Abhishek Santra

By Team 5

mohan.deshpande (1001651135)
sangramsinha.patil (1001696509)

Table of Contents

Description

This project implements parts of the index file organization for the database management system MINIBASE. Our aim is to implement the full insert and a naive delete (simply remove the record without performing any merging or redistribution) algorithm. We will implement a B+ tree in which leaf level pages contain entries of the form <key, rid of a data record> (Alternative 2 for data entries, as discussed in the textbook on page 276.)

Overall Status

We have successfully implemented full insert() and NaiveDelete() method. Here is brief overview of how major components were implemented.

Function insert()
Before we begin inserting first record, we have an empty B+ tree. From BTTest.java, we start with function runAllTests() which calls this function for specified amount of records. It is therefore important to know whether B+ tree is initialized or not. In this function we create first root-leaf page and insert record for first time. We update header to point to new root page by using updateHeader(newRootPageId).

Once the B+ tree initialized we call _insert() which handles subsequent addition of records. Whenever there is new root e.g. while inserting 63rd entry (new leaf root) or 2635th entry (new index root), we need to update the header, this is handled by checking if(newRootEntry != null).

Function _insert()
This is the core function of this project. Algorithm mentioned in Project1 Demo_v1.pdf is used primarily for implementation.

To begin with we associate the sorted page instance with the page instance. This is primarily do identify if the page is INDEX or LEAF by using function .getType() == NodeType.INDEX/LEAF

if LEAF page is encountered –
Associate the BTLeafPage instance with the Page instance. And Insert record if available page size > record size. If space is not available we must create new BTLeafPage(headerPage.get_keyType()), handle the split and manage doubly link list.
Once the new Leaf is created here is how split is managed.

1. Cut all records from current page and paste to new page (strategy mentioned in Project1 Demo_v1.pdf) by using currentLeafPage.getFirst(delRID).
2. Cut half records from new page and paste to current page by using newLeafPage.getFirst(delRID). Same as above, only difference is top entries from new page are copied to current page. Repeat this by comparing space between newLeafPage.available_space() and currentLeafPage.available_space()
3. Handle current record after split by using BT.keyCompare. We will compare last record transferred from new page to current page in previous step.

if INDEX page is encountered –
While inserting records our aim is to reach LEAF page. If we have arrived at INDEX page it means we need to traverse. To traverse to next node we need to know the next page. Next page is identified by calling function currentIndexPage.getPageNoByKey(key) and then we recursively call _insert(key, rid, nextPageId) until we reach LEAF. If LEAF split has occurred which is tracked by (upEntry == null) we will insert entry into INDEX.

To insert entry into INDEX page, we follow all the same steps as done in insert of LEAF.

NaiveDelete()
Idea behind naive delete is to traverse to the leftmost leaf first. We achieve this by calling findRunStart(key, currentRID).

Once we have reached to the leftmost leaf, our aim is to traverse to right newLeafPage.getNextPage() and continue the same until we find the page where we have a record.

Once we are on the page where record exists, we call newLeafPage.delEntry(new KeyDataEntry(key, rid)). This will return true/false depending upon the successful deletion.

To manage delete of duplicates we have maintained variable int flag. Delete function is called recursively until it returns false, therefore confirming delete of duplicates. If any deleting happens on page successfully we should write to disk by unpinPage(newLeafPage.getCurPage()); and stop traversal immediately.

Delete() (Bonus)
Due to timeline constraints, we started with full delete but were not able to finish implementation and test it within submission deadline. Therefore, this is not included in project submission.

File Descriptions

We don't have new files created for this project. Implementation of insert(), _insert() and NaiveDelete() functions was added to BTreeFile.java file.

Major functions

insert() – This function initializes B+ tree by creating root page. We also call _insert() function to actually insert records from this function. This function keeps track of new root created and updates header.
_insert() – This function handles the core logic of this project. Node traversal, record insertion to leaf pages, copy up/push up, index split, leaf split is managed through this function.
NaiveDelete() – This function handles the naive delete i.e. simply remove the record without performing any merging or redistribution.

Additional test cases

- TestCase: Insert same record multiple times and delete same record.
  Result: All occurrences of duplicate record should get deleted.
  E.g. insert value '10' three times, then delete 10. This should delete all 3 occurrences of 10.

- TestCase Insert 4000 entries.
  Result: Height of tree should be 3.
  e.g. After 2634th record index split required, i.e. while inserting 2635. 1000/12*32 = 2666 - 32 = 2634. Therefore by inserting 4000 entries this condition is met and hence this tests if index split is managed properly or not.

Division of Labor

Mohan Deshpande – Implementation of Index Split and Naive Delete.
Sangramsinha Patil – Implementation of Leaf Split and handling delete of duplicate records in Naive delete.

Logical errors and how you handled them

1) When we started to implement the project, we followed Project1 Demo_v1.pdf and aimed to finish in one go. But as we printed the tree after writing most of the code, results were not good. Only partial tree was getting printed. To rectify we only focused only on LEAF insertion first and commented other functionality. We identified we were using wrong constructor BTLeafPage currentLeafPage = new BTLeafPage(headerPage.get_keyType());
This was very difficult to identify, as there was no error thrown while execution and still only partial tree was printed. After couple of discussions we had a look at existing provided function and used correct constructor as below
BTLeafPage currentLeafPage = new BTLeafPage(currentPage,headerPage.get_keyType());

2) Handling the leaf split was very difficult initially. Not being familiar with MINIBASE api made us to do lot of dry run. While transferring record we thought of .available_space() and comparing it with manual value 512 (i.e. 1024/2 = 512 bytes) to stop the transfer. But the split was inconsistent as observed during subsequent inserts. We also faced below error often while managing splits -

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!        Something is wrong                !!
!!    Is your DB full? then exit. rerun it!    !!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
java.lang.NullPointerException
        at btree.BTreeFile._insert(BTreeFile.java:511)
        at btree.BTreeFile.insert(BTreeFile.java:380)
        at tests.BTDriver.runAllTests(BTTest.java:159)
        at tests.BTDriver.runTests(BTTest.java:79)
        at tests.BTTest.main(BTTest.java:567)
```

We then tracked (debug) available space for newLeafPage and currentLeafPage and arrived at conclusion that using newLeafPageFreeSpace < currentLeafPageFreeSpace as limit condition in loop is more appropriate.

3) First entry added in tree is added twice while creating root page. To handle this we added return statement in if(pageno.pid==INVALID_PAGE) loop.

4) During implementation of NaiveDelete we encountered error –

```
bufmgr.PageUnpinnedException: BUFMGR: PAGE_NOT_PINNED.
        at bufmgr.Replacer.unpin(Replacer.java:66)
        at bufmgr.BufMgr.unpinPage(BufMgr.java:611)
        at btree.BTreeFile.unpinPage(BTreeFile.java:107)
        at btree.BTreeFile.NaiveDelete(BTreeFile.java:773)
        at btree.BTreeFile.Delete(BTreeFile.java:587)
        at tests.BTDriver.runAllTests(BTTest.java:221)
        at tests.BTDriver.runTests(BTTest.java:79)
        at tests.BTTest.main(BTTest.java:567)
bufmgr.PageUnpinnedException: BUFMGR: PAGE_NOT_PINNED.
        at bufmgr.Replacer.unpin(Replacer.java:66)
        at bufmgr.BufMgr.unpinPage(BufMgr.java:611)
        at btree.BTreeFile.unpinPage(BTreeFile.java:107)
        at btree.BTreeFile.NaiveDelete(BTreeFile.java:773)
        at btree.BTreeFile.Delete(BTreeFile.java:587)
        at tests.BTDriver.runAllTests(BTTest.java:221)
        at tests.BTDriver.runTests(BTTest.java:79)
        at tests.BTTest.main(BTTest.java:567)
```

After a little debug we identified that even if we were able to manage delete (even for duplicate entries), loop continued. Once we delete the record we use unpinPage. And if we let loop continue unpinPage is called twice for same page. This creates above error. We identified that after performing the delete and unpinPage, we should break out of loop. A return true statement solved this.

Conclusion

Unfamiliarity with MINIBASE API presented us with lot of surprises while implementation. Though with constant guidance of GTA Abhishek Santra and professor Sharma Chakravarthy, we were able to get through the initial obstacles. We followed industry coding standards while implementing this project. Being our first project for this course, we identified that we need better planning in future, to go for Bonus points. Our learning from this project will help us in achieve higher in upcoming projects.

References

1) https://wweb.uta.edu/faculty/sharmac/javadocs/index.html
2) https://wweb.uta.edu/faculty/sharmac/mini_doc-2.0/minibase.html
3) http://www.eecs.yorku.ca/course_archive/2013-14/W/4411/proj/javadoc/ (for Btree package)
4) https://elearn.uta.edu/bbcswebdav/pid-7571195-dt-content-rid-133529008_2/xid-133529008_2
(Project1 Demo_v1.pdf for pseudo code.)