Project 1: B+ tree implementation

Jay D. Bodra
ITLAB@UTA
Email: jay.bodra@mavs.uta.edu
URL: itlab.uta.edu

Talk Outline

- Understanding B+ tree concept in minibase
- Algorithm for insert(), _insert(), NaiveDelete()
- Demo

Understanding B+ tree concept in minibase

- HeaderPage
  - Header Page is the root node
  - There is only one Header Page
  - When you run the program a header page is created by the BtreeFile(String, int, int, int) constructor which will be pointing to an INVALID_PAGE

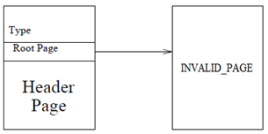
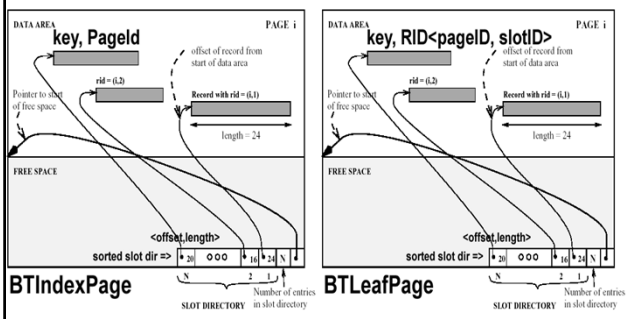


Figure 1: Initial B+ tree headerPage points to an INVALID_PAGE

Understanding B+ tree concept in minibase(1)

- IndexPage
  - Index Page is represented in the form of <key, PageId>
  - It points to the left leaf Page by its pageId and the rest leafPages that it points to are stored in form of <key, PageId>
- LeafPage
  - Leaf Page is represented in the form of <key, PageNo, SlotNo>
  - Leaf Pages are organized as a doubly link list
  - Previous pointer of Left most leaf page points to INVALID_PAGE
  - Next Pointer of the right most leaf page points to INVALID_PAGE

Understanding B+ tree concept in minibase(2)



An overview of given code

- To create a new Leaf Page or IndexPage
  - BTLeafPage newRootPage=new BTLeafPage(headerPage.get_keyType()); defined under BTLeafPage class
  - What will the code for IndexPage?
- PageId newRootPageId = newRootPage.getCurPage();
  - getCurPage() is a method defined under HFPage Class which returns the page number of current page
- currentPage.getType() == NodeType.***INDEX***
  - getType() is a method defined under HFPage Class which return the type of page – leaf or Index
- nextPageId=currentIndexPage.getPageNoByKey(key)
  - getPageNoByKey is a method defined under IndexPage which will return the pageId or pageno of the page holding the key that is passed

Algorithm for insert()

- Check the headerPage.get_rootId().pid == INVALID_PAGE
  - the tree is empty and we have to create a new first page **newRootPage**; this page will be a **leaf page**
  - set the newRootPage.setNextPage(new PageId(INVALID_PAGE)) similarly set the PrevPage pointers of the new page created as it is a leaf node
  - insert the record on the page that is created
  - unPinpage the newRootPage as it is dirty [Note: it is help when the lower index page or leaf page gets spilt]
  - update the header updateHeader(newRootPage)
- Now check for a split
  - Create an instance of KeyDataEntry newRootEntry that will to catch the return statement from _insert(KeyClass, RID, pageId) method

Algorithm for insert() cont..

- If the newRootEntry is not null means a spilt occurred
  - Create a new page; this time the page will be an Index page as the initial leaf page got spilt
  - Insert record on this index page in the form of <key, pageId>; newRootPage.insertKey(newRootEntry.key, ((IndexData)newRootEntry.data).getData())
  - The old root is split and it will now become the left child of new root; set the prevPage pointer to the old root using headerPage.get_rootId()
  - unPinpage the new root using its page id
  - Update the header to new root using its page id

Algorithm for _insert()

- Initially create an empty Page page; and pin it using the pageId passed to the _insert() and then create a BTSortedPage currentpage of the page which will associate the sorted page instance with the page instance; Also create a KeyDataEntry upEntry
- Check if currentpage is of type Index
  - Create a BTIndexpage currentIndexPage, a variable to store its pageId CurrentIndexpageId, and a variable to store the pageId of the new key nextPageId=currentIndexPage.getPageNoByKey(key)
  - Unpin the currentIndexPage using its pageId; recurse by using upEntry and passing correct parameters to _insert() then pin it again
    - If upEntry is null no split occurred; return null;
  - Check if the currentIndexPage has space for new entries currentIndexPage.available_space() >= BT.getKeyDataLength(upEntry.key, NodeType.INDEX)
    - If true insert the data on currentIndexPage and unpin the page as it is dirty
  - No space is available so create a newIndexPage; get its pageId

Algorithm for _insert() contd..

- Create a tmpkeyDataEntry and a RID delRID = new RID() , Use a for loop for (tmpEntry= currentIndexPage.getFirst(delRid); tmpEntry!=null;tmpEntry= currentIndexPage.getFirst(delRid)) to transfer all the records from currentIndexPage to newIndexPage
  - Insert the records on newIndexPage
  - Delete records from currentIndexPage=deleteSortedRecord(delRid)
- Make the split equal using other for loop to spilt the records equally[hint: use a if statement to undo last record]
- Compare the key using BT.keyCompare(upEntry.key, tmpEntry.key)
  - If the value is positive the new key upEntry.key goes on the newIndexPage
  - Else on the currentIndexPage

Algorithm for _insert() contd..

- Unpin the currentIndexPage as it is dirty
- Fill up the upEntry= newIndexPage.getFirst(delRid)
- Set the left link in the newIndexPage .i.e. setPrevPage [hint: pass the upEntry.data]
- Delete the first record from newIndexPage
- Unpin the newIndexPage as it is dirty
- Set the higher Index page in the hierarchy to point to the newIndexPage; ((IndexData)upEntry.data).setData(newIndexPageId)
- Return upEntry

Algorithm for _insert() contd..

- Else check if currentpage is of type Leaf
  - Create a BTLeafpage currentLeafPage, a variable to store its pageId CurrentLeafpageId
  - Check if the currentLeafPage has space for new entries currentLeafPage.available_space() >= BT.getKeyDataLength(upEntry.key, NodeType.LEAF)
    - If true insert the data on currentLeafPage and unpin the page as it is dirty
  - No space is available so create a newLeafPage; get its pageId; set the nextPage and prevPage pointer for newLeafpage and nextPage Pointer for CurrentLeafPage
  - Create a tmpkeyDataEntry and a RID delRID = new RID() , Use a for loop for (tmpEntry= currentLeafPage.getFirst(delRid); tmpEntry!=null;tmpEntry= currentLeafPage.getFirst(delRid)) to transfer all the records from currentLeafPage to newLeafPage
    - Insert the records on newLeafPage
    - Delete records from currentLeafPage=deleteSortedRecord(delRid)

Algorithm for _insert() contd..

- Make the split equal using other for loop to spilt the records equally [hint: use Keycompare to undo the last record]
- Compare the key using BT.keyCompare(key, undoEntry.key)
  - If the value is positive the key goes on the newLeafPage
  - Else on the currentLeafPage
- Unpin the currentLeafPage as it is dirty
- Fill up the tmpEntry= newLeafPage.getFirst(delRid); upEntry=new KeyDataEntry(tmpEntry.key, newLeafPageId)
- Unpin the newLeafPage as it is dirty
- Return upEntry
- Else throw and insertException(null, “”)

Algorithm for NaiveDelete()

- Create a leafPage; a iterator of type RID and a KeyDataEntry entry
- Use the function findrunStart(KeyClass, RID) to find the first page and rid of keys
- If leafpage is NULL return false
- Set entry = leafPage.getCurrent(curRid)

Algorithm for NaiveDelete() contd..

- While(true)
  - While(entry == null)
    - Have to go right; nextpage = leafPage.getNextPage()
    - Unpin the previousPage
    - if nextpage.pid == INVALID_PAGE; return FALSE
    - Initialize the leafPage to nextPage along with pinning the nextPage
    - Initilize entry by get the first RID of the leafPage
  - Using KeyCompare check the key and entry.key; if positive value break
  - Check leafPage.delEntry(new KeyDataEntry(key, rid)) == true; the key is successfully found and is delete; Unpin the leafPage as it is dirty
  - Go right nextpage = leafPage.getNextPage()
  - Unpin the leafPage using getCurPage()
  - Initialize the leafPage to nextPage along with pinning the nextPage
  - Initilize entry by get the first RID of the leafPage
- Unpin the leafPage using getCurPage()
- Return false

Thank You !!!