Project 4: B+ Tree Index Manager
Design Report

Miao Yang, Qimeng Han, Group 8

Changes in btree.h:

1. Added int numKeys to the struct NonLeafNodeInt{ } and struct LeafNodeInt{ }
2. added int level to struct NonLeafNodeInt{ }.
3. Initialized the bool scanExecuting to false.
4. Adjust the INTARRAYLEAFSIZE and INTARRAYNONLEAFSIZE by minus the size of int level.
5. Deleted the unused variables. Including : double lowValDouble; std::stringlowValString; double highValDouble; std::string lowValString; std::string highValString; Datatype attributeType;

Added additional test cases in main.cpp:

We have added the following additional test in the void intTests() method

checkPassFail(intScan(&index,-2008,GT,2008,LT),2008)

use the larger negative for lowVal and larger positive value for highVal

checkPassFail(intScan(&index, -100,GT, -1,LT),0)

use the negative values for both lowVal and highVal

checkPassFail(intScan(&index,1000,GTE,4000,LT), 3000)

Use to test the ability of scan the larger number of records

BTreeIndex

The constructor first checks if the corresponding index file exists by checking the outIndex name. If the specified index file exists, read the first page from the file (meta page), cast the metaPage into metaNode, and then get the root page number from the meta node. Unpin any pages that have been read. Else if the specified index file does not exist, create a new index file. Creating a new index file has a few steps: first, allocate a new meta page, cast into meta node, allocate new root page, then initialize the information in the index. Unpin any pages that have been read. In the end, scan the record and insert into the B+ tree.

~BTreeIndex

The destructor. Flushing the index file, delete file instance thereby close the index file.

insertEntry

We recursively insert the entry by calling the insert method on the appropriate child node. The algorithm traverses from the root node to the leaf node, placing the entry there, and then returning all the way back to the root node. If the old root is split, create a new root, the height of the tree increases by 1 (the root level increments by 1). So in this case, the level of leaf node is 0, then increment the level of the node, for example, the level of node just above leaf node is 0. Unpin any pages that have been read.

insert

We consider two different cases for insert. One case is to insert leaf node. Another one is to insert nonLeaf node.

insert leaf node:

if the leaf node is not full, just call another insertNonFullNode method to directly insert key/rid pair into the node, and increment the number of keys in the node. **If the leaf node is full,** it must be split. To split the node, we first create a new leaf node, and move the second half of the keys/rids from the current leaf node to the new leaf node, then compare the key (from entry) to the middle key of the leaf node, if the key less than the middle key, add it to the current leaf node, otherwise, add it to the new leaf node. Return the pageNo and unpin any page that have been pinned.

insert nonLeaf node:

First, we find the appropriate child node to insert by comparing the key value (which will be use to insert) to keys in the current node. Then get the target child node pageNo. Based on this child pageNo, recursively insert into the child. **If the current non leaf node is not full**, directly insert the key/pageNo, increment the number of keys of the current non leaf node. **If the current node is full,** split the node. First, create a new non leaf node. Move half of the key to the new non leaf node. Compare the key value to the middle key value, decide which non leaf node to insert. If the key less than the middle key, insert into the current non leaf node, otherwise insert into the new non leaf node. Unpin any pages that have been used and return the pageNo.

insertNonFullNode

The insertNonFullNode is used to insert the entry (key/rid pair for leaf node, key/pageNo for non leaf node) to the node that is not full. The method deals with two cases: 1. Insert into nonfull leaf node. 2. Insert into nonfull nonleaf node

Insert into nonfull leaf node:

First, find the position to insert by comparing the key (use to insert) to the keys in the current leaf node. Then move over the key/rid array to right and allow the key/rid to insert. Insert the key/rid to the right position in the array.

Insert into nonfull nonleaf node:

Similar idea but a little difference comparing to the leaf node. Insert pageNo to the nonleaf node. The number of pageNo = number of keys +1.

Similarly, find the position to insert by comparing the key (use to insert) to the keys in the current nonleaf node. Then move over the key/pageNo array to right and allow the key/pageNo to insert. The position of pageNo = The position of key + 1. Insert the key/pageNo to the right position.

startScan

We first check whether the operators are correct, whether another scan is in executing and if the lower value is greater smaller than the larger value. If all these conditions are satisfied, then we star the scan starting from the root to find which page does the lowValParm is on. We have separated this method into two parts, one part is the scan through nonLeafNode, which has three conditions, the lowValParm is less than the smallest value in the nonLeafNode; larger than the largest value in the nonLeafNode or in between. This part will continue to run until we have reached the leaf node. Once reach the leaf node, similar comparison will be conducted to find which page the lowValParm is on. We use a boolean value “found” to tack whether the lowValParm has been founded yet. The while loop will continue running until the lowValParm has been found.

scanNext

First check if there is a currently executing scan, if not throw an exception. If the scan has already completed, throw exception. If there is a currently conducting scan, then check if the index of the

next entry is in the range of the size of the array, if it is, then see if the next key value is less the highValParm, if not, throw an exception, if yes, update the outRid to the next entry on this node. If all the keys in the current page has been scanned, then move to the next page.

endScan

Terminates the current scan and unpins all the pages that have been pinned for the purpose of the scan. If not scan has been initialized, throw an exception.

Handle duplicate keys

To retrieve all data entries with a given key if there are duplicates, we need to find the left-most data entry and then possibly retrieve more than one leaf pages. To find the left-most position to insert, we need to modify the find position method in our insert algorithm. Using key value is less or equal to the given key value to find the left-most of the entries on an index page. We need to search to the left using a neighbor pointer at the leaf level until find the key value is less than the given key value (k). Then, scan forward along the leaf level to find a position with a key greater than k.

To find the position of the data entry in the b+ tree, we can also consider to use rid value in the data entry to be part to of the search key, which will make the index into a unique index.

If there are large number of duplicates, a single data entry can lead to multiple pages. To address that, we can maintain the list of rids with each data entry in sorted order.

UpinPages

Unpin the page after finishing the reading or allocate the page.

During range searches, we did not find unnecessarily traversing the tree up and down.