Design Report

Group 7
Team members

Name	NetID
Qian Zhang	qzhang348
Lin Su	lsu32
Tina Xiong	jxiong49

The implementation of B+Tree is as follows.
Every time we are done using a page, we will unpin that page to prevent exception like HashAlreadyExistException.
This implementation is very efficient, there is no redundant operation. The running time of test1, test2, test3 and test4 designed by ourselves are about 0.3s respectively. And the errorTests also passed.

Insert Entry:

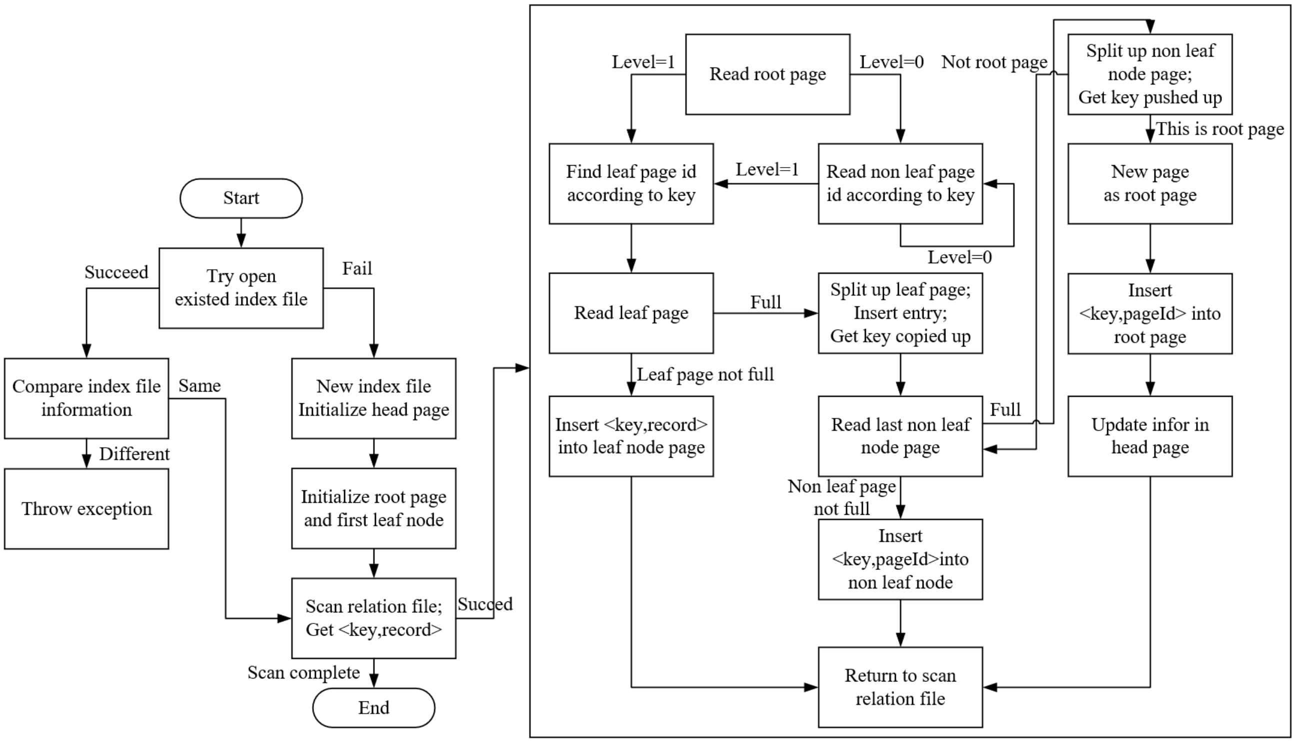
Start
Try open existed index file
Succeed
Fail
Compare index file information
Same
Different
Throw exception
New index file Initialize head page
Initialize root page and first leaf node
Scan relation file; Get <key,record>
Succed
Scan complete
End
Read root page
Level=1
Level=0
Not root page
Find leaf page id according to key
Level=1
Read non leaf page id according to key
Level=0
Read leaf page
Full
Split up leaf page; Insert entry; Get key copied up
Leaf page not full
Insert <key,record> into leaf node page
Read last non leaf node page
Full
Non leaf page not full
Insert <key,pageId>into non leaf node
Split up non leaf node page; Get key pushed up
This is root page
New page as root page
Insert <key,pageId> into root page
Update infor in head page
Return to scan relation file

Figure 1: Index file initialization and insertion of entry from relation file

Notes:
When we traverse down the tree to find leaf node page to insert data entry, we record the page ids so it’s convenient for us to traverse up the tree if we need to split up leaf node and non leaf node.

Start Scan:

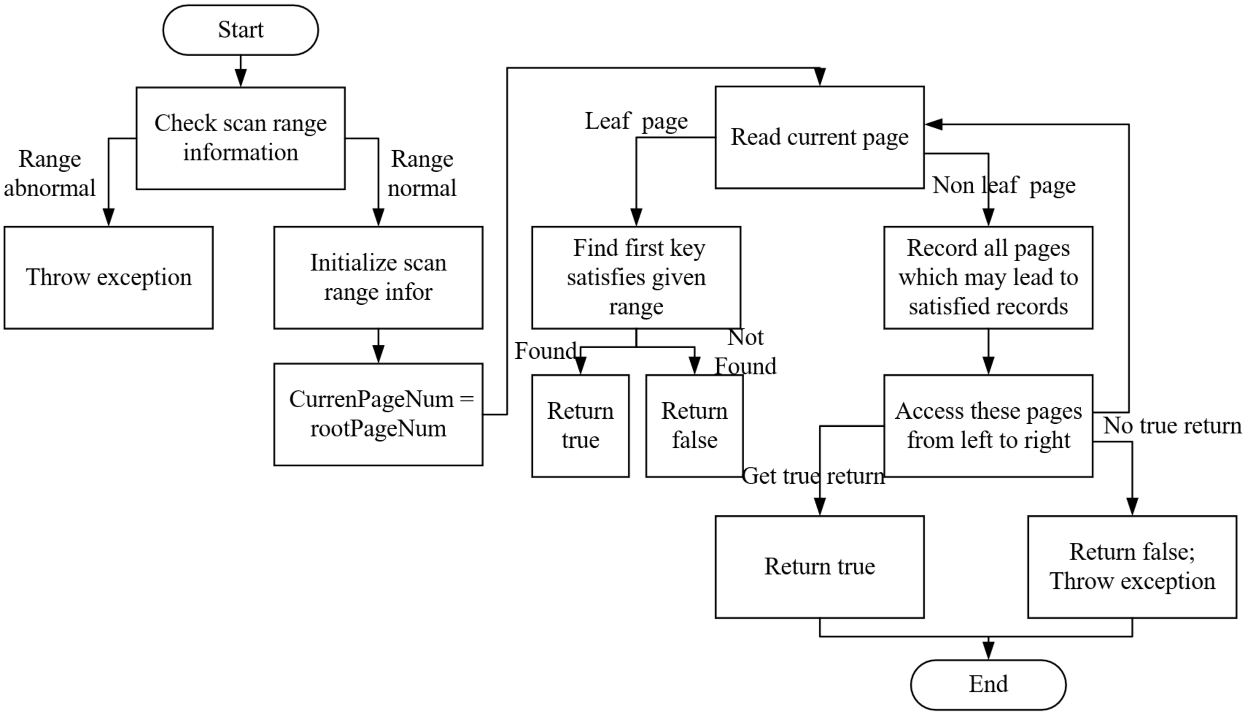
Start
Check scan range information
Range abnormal
Range normal
Throw exception
Initialize scan range infor
CurrenPageNum = rootPageNum
Read current page
Leaf page
Non leaf page
Find first key satisfies given range
Found
Not Found
Return true
Return false
Record all pages which may lead to satisfied records
Access these pages from left to right
No true return
Get true return
Return true
Return false; Throw exception
End

Figure 2: Scan initialization and find first entry in leaf node page

Scan Next:

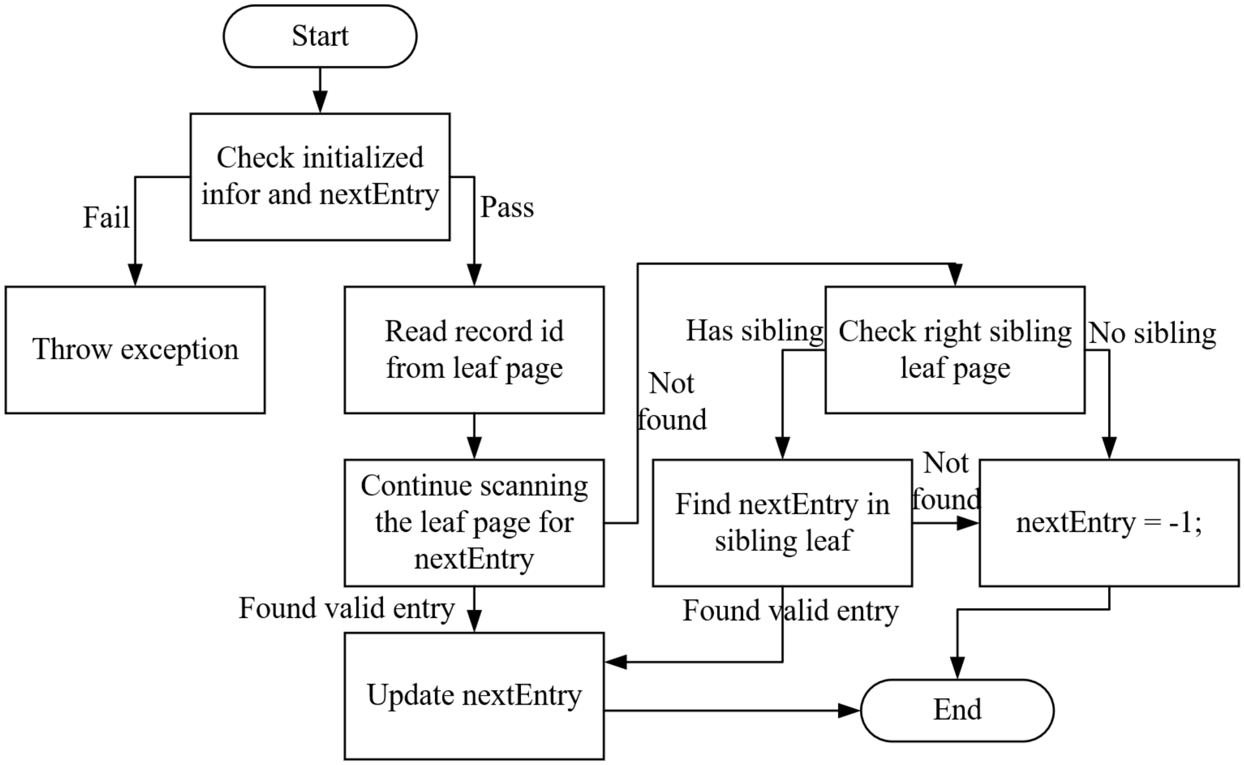
Start
Check initialized infor and nextEntry
Fail
Pass
Throw exception
Read record id from leaf page
Continue scanning the leaf page for nextEntry
Not found
Check right sibling leaf page
Has sibling
No sibling
Find nextEntry in sibling leaf
Not found
nextEntry = -1;
Found valid entry
Found valid entry
Update nextEntry
End

Figure 3: Find next entry satisfies given range

If duplicated keys are allowed:

The simplest design is to have multiple entries with same key in the tree. But when the entries are too many, it’s not very efficient to find these entries.
The second design is to change the data entry in leaf node as <key, pointer>, the pointer points to a linked list which points to all records with that key.

Tests Design:

(1) Construct a relation with sparse values of key {0,2,4,6,8….,2498}, and then construct tree with this relation and then implement the same test as test1, test2, test3.
And all of tests passed.
Check test4(), createRelationSparse(), indexTests_Sparse(), intTests_Sparse() in main.cpp for code details.
(2) Based on test1(), test2(), test3(), and test4() mentioned above, search key from -500, 5500 in these scenarios, and the tests also passed.
Check intTests(), intTests_Sparse() in main.cpp for code details.