CSE 5331 DBMS MODELS AND IMPLEMENTATION

10/1/2015

B+ Tree Project

Full Deletion

Rajaraman Govindasamy
1001165700
Team 5

Contents

Core Concept 2
Overall Status 2
File descriptions 3
Division of Labor 3
Logical Errors 3

Core Concept

The project mainly focus on implementing the full deletion in a B+ Tree structure. B+ trees contain multiple entries, each containing a key value and a left and right child. Leaf nodes contain the data and the non-leaf nodes contains the indexes. The adjacent sibling nodes are linked together with right and left sibling pointers. The main focus area to implement is the full deletion of the B+ tree attempting to delete the key from a leaf page that is less than half full should cause the page either to redistribute or merge with one of the siblings. The minimum occupancy is taken into consideration and is kept to default of 50 percent to avoid confusions. If one of the page's sibling has any space to spare in the page then the keys have to be evenly distributed between the pages, and the parent page should be updated accordingly. Deleting entries from the parent will make it less than half full. In this case, the same parent should try to redistribute itself or merge with its sibling. This scenario will lead to multiple recursive merges and deletions which need to be handled efficiently as part of the system.

Overall Status

My basic approach towards this problem was top down starting from the analysis of the given base code which was needed to be in sync with the nomenclature and understand the coding complexity of the existing system. I started with the implementation of the simple or the direct delete without the siblings present as the first scenario in the B+ tree. The *delEntry ()* method was used for the same and implemented the other possible scenarios where the tree has only the root node and when the tree is empty.

The next level was to implement the deletion with the siblings included which was the difficult part or the core part of the full deletion algorithm. The main focus here was to implement the code to redistribute and merge the keys between the leaf pages. This included the checking the available spaces using the ***available_space ()*** method along with the slot size to accommodate the new key values in line with the minimum occupancy level, re arranging of the key and the leaf pages on merge or distribution, updating the parent pointers when the child are moved to the adjacent siblings. The ***redistribute ()*** method in the BTLeafPage class was used to perform the redistribution function. The merging of leaf pages are done when there is no possibility of redistribution, followed by the re arrangement of the leaf pages. The next level of iteration of the same is done when the search key is not returned in the initial search of the index pages. The proper pinning and unpinning of the pages were the tortuous part to get the algorithm proper. The **pinPage()** and **unpinPage ()** methods of the BTreeFile class was used for the same.

File descriptions

No new files were created.

Division of Labor

NA

Logical Errors

Below are few logical errors that I handled as part of the full deletion implementation of the B+ Tree.

1. *Improper updating in the leaf nodes*

In the deletion case, the key values were inserted in a wide range and I chose a leaf page and deleted the set of entries till the minimum occupancy level in that particular page. The remaining entries were merged to the sibling page but the leaf pages were not updated properly by deleting the current page from the leaf node page. This was misleading when displaying active leaf pages.

Solution:

This behavior was due to the improper assignment of the right and left child during the merge operation. After proper assignment of the right and left child to sibling and leaf pages the issue was resolved.

2. *Minimum occupancy*

The minimum occupancy implementation was having issues at the initial stage where the occupancy level didn't trigger the redistribution nor the merging of the entries to the siblings rather were intact to any level of deletion. Even when the page was way less than the minimum occupancy level there was no use.

Solution

The alteration and the inclusion of the proper slot size of the block and page the same was rectified and the functions started to respond on merge or redistribute reaching the minimum occupancy level.

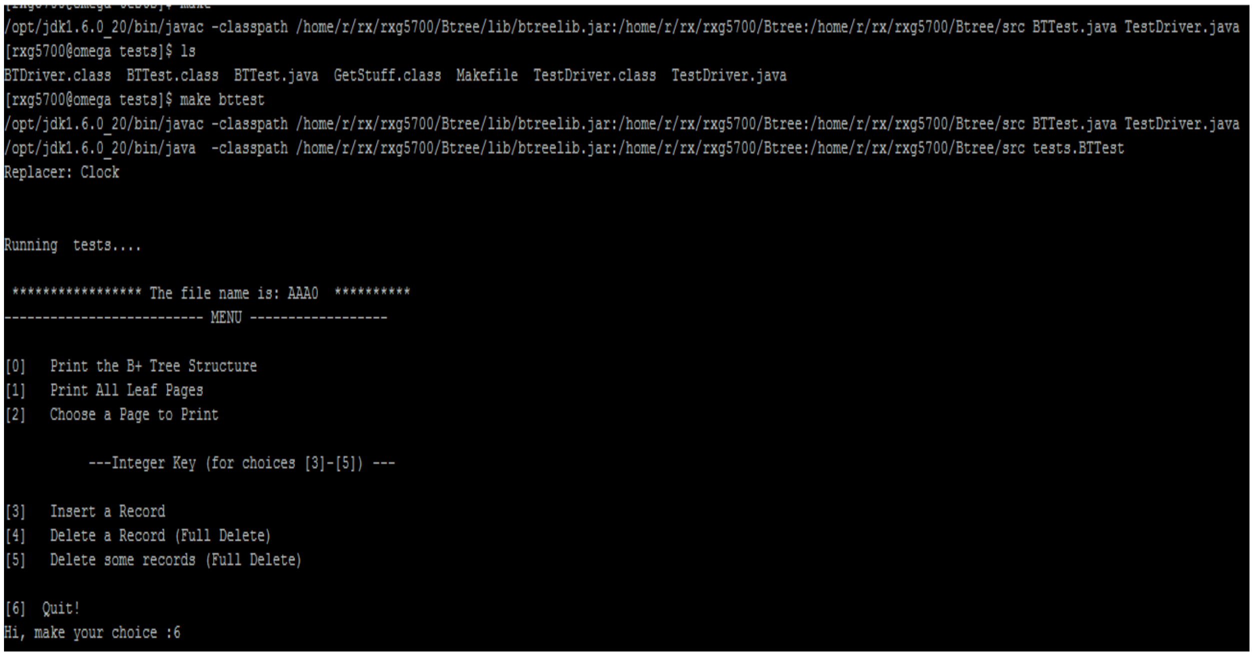
3. *Unpin and Pin page Issues*

The Unpin and Pinning of the pages was not handled properly at the initial stage of the implementation phase. The Page not pinned and the page pinned exceptions were popping frequently and was very challenging to handle.

Solution

After including these functions at required places inside the functions the code started responding properly.

Successful execution output:



```
/opt/jdk1.6.0_20/bin/javac -classpath /home/r/rx/rxg5700/Btree/lib/btreelib.jar:/home/r/rx/rxg5700/Btree:/home/r/rx/rxg5700/Btree/src BTTest.java TestDriver.java
[rxg5700@omega tests]$ ls
BTDriver.class  BTTest.class  BTTest.java  GetStuff.class  Makefile  TestDriver.class  TestDriver.java
[rxg5700@omega tests]$ make bttest
/opt/jdk1.6.0_20/bin/javac -classpath /home/r/rx/rxg5700/Btree/lib/btreelib.jar:/home/r/rx/rxg5700/Btree:/home/r/rx/rxg5700/Btree/src BTTest.java TestDriver.java
/opt/jdk1.6.0_20/bin/java  -classpath /home/r/rx/rxg5700/Btree/lib/btreelib.jar:/home/r/rx/rxg5700/Btree:/home/r/rx/rxg5700/Btree/src tests.BTTest
Replacer: Clock


Running  tests....

 **************** The file name is: AAA0  **********
------------------------- MENU ------------------

[0]   Print the B+ Tree Structure
[1]   Print All Leaf Pages
[2]   Choose a Page to Print

           ---Integer Key (for choices [3]-[5]) ---

[3]   Insert a Record
[4]   Delete a Record (Full Delete)
[5]   Delete some records (Full Delete)

[6]  Quit!
Hi, make your choice :6
```