|  |
| --- |
|  |

|  |
| --- |
| **B+ TREE IMPLEMENTATION USING MINIBASE** |
|  |
|  |
| This project implements inserts and naïve delete of the index file organization for the database management system using MINIBASE |
|  |
| Submitted By: |
| **Akshay Sarkar - 1001506793 Remesh Sreemoolam Venkitachalam – 1001414827** |
|  |

**Table of Contents**

Table of Contents

**Overall Status2**

Insert Implementation2

Delete Implementation3

**File Description 4**

**Division of Labor4**

**Logical Errors Faced 4**

**B+ Tree Project Report**

**Overall Status**

B+ Tree indexing is a popular index technique used for retrieval of data. In this project, we create modules that deal with the ‘insertion’ and ‘deletion (naive delete)’ of nodes in the B+ Tree index.

We began by understanding the structure and implementation of Minibase library as provided on the class website. With the help of Professor and TA (Teaching Assistant), we got illuminated with the structure of a B+ Tree and how we could implement a simple B+ Tree. We researched more along with the helpful documentations provided by our TA and planned the algorithm for both insertion and deletion features.

We have successfully implemented a full insert using the functions ‘insert()’ and ‘\_insert()’, and naive delete using function ‘delete()’ in B+ Tree Structure. We came across several bugs which are explained in detail below, but worked around it and cleaned it up to have our algorithm execute without flaws.

* **Insert Implementation**

In this feature implementation, we started with insert () method which takes the parameters <key, recordID>. Initially, B+ Tree would be empty and header would be pointing to an INVALID\_PAGE. We create a new leaf page and insert the <key, recordID> value into it. We also update the header to point it to newly created leaf page.

All the further insertion would happen via \_insert () function which takes parameters as <key, recordID, currentPageID>. The \_insert() function checks whether the arriving page is a Leaf or an Index. In case, it is a leaf page, it checks for space in the page and inserts into it. Once the leaf page becomes full, we need to split the data into a newly created leaf page, and generate an index page for mapping up the data values. The records in the first leaf page is counted and then it is traversed to reach the middle record. From this point on the entries are copied into the new leaf and deleted from the old one, such that both leaf pages are half full. The new key is then inserted into the page where it belongs in sorted fashion. We also need to copy up the first ID into the index above, which if not present at the very first leaf split, has to be created and made to point to the two leaf pages, and also establish itself as new root by getting the header page to point to it. This process continues until the index is full.

In case the index is full, we split the index by copying all entries into new leaf and then rewriting the first half back into the old index. This will cause to move up the first entry of the new index into the upper hierarchy. If the root was split, then we create a new root and move up the first entry into new root and point the header page to new root, while directing the left pointer to old root and a pointer to new split index.

So as the index builds up, when you insert a new record, the control starts at the root, and comes down the appropriate index nodes, and finally reaches the correct leaf node where the insertion should happen. In case Insertion causes a split it is accordingly updated in the index nodes.

Duplicates are allowed in this index tree, and are placed adjacent to each other in sorted fashion.

* **Delete Implementation**

In this feature, we implemented naiveDelete method which takes the parameters <key, recordID>. We traverse the tree using key comparison to find the key which has to be deleted using the findStartRun(). Once we locate the entry it is deleted using delEntry(). It is a naive implementation where the redistribution or merging of nodes of tree after the deletion are not being checked.

In case of multiple key delete, we require horizontal traversal of the leaf nodes, by finding the nextID. When there are more keys to be deleted, but we reach the end of the present leaf node, we traverse to the next Leaf page node to find the rest of the keys.

In case of duplicates, only the first entry was being deleted. But as per a previous mail from the TA asking us to delete all occurrences of the key, we modified the implementation to do the same for single key delete and deletion of a range of keys.

When the range spanned multiple leaf nodes, we found that just putting a loop on deletion is not enough. When the entry record reaches null, we need to traverse to next leaf node and check if those contain the same key values. If they do, those entries should be deleted too. We also set a flag ‘deleted’ to indicate if any records have been deleted or not.

**File Description**

No additional files were created for our project. All the files we have used were provided with the initial bundle.

**Division of Labour**

We took a couple of meetings together to understand the B+ Tree structure and study the functions and variables given to us. Figuring out the use of various functions and how we will be able to use these functions and constants in our algorithm was a daunting task which was divided among both of us. We identified some important classes provided on the java doc and divided the work of understanding the structures of these classes.

Once we accustomed ourselves with the classes, we met to discuss about the research and create a plan to formulate the algorithm.

Since the working of the insert and delete functions were dependent on a sequential manner of inputs and its distribution, we decided to tackle insert first. We followed the pdf given in class for direction and started working on the code. We delegated the work in the following manner where respective persons took charge of the below given functions:

Remesh - \_insert(Leaf) and naiveDelete()

Akshay - insert() and \_insert(Index)

By coding together, we were able to help each other with the debugging of the program, and that made it easier to find flaws in the code.

**Encountered Errors**

1) When Copying data from one leaf to another, the algorithm that we need to set up there was a challenge. How to copy the content from certain position and remove it from the first leaf.

The current insert algorithm split the old leaf into two, but was not inserting all the necessary records into the new leaf. Many records were missing.

We found it was due to the use of getNext() which skipped every other entry, and changed it to getCurrent().

2) Wrong assumption that the right most leaf only splits made us to think that the new split leaf need to have a next page set to null. This started causing errors when duplicate keys started being inserted. When deletion of a range of values that spanned multiple leaf nodes was introduced, it ended at some leaf nodes even when there were more keys to be deleted. On scrutiny we found that we are not connecting the right pointer or end pointer to next leaf in some splits but is made null. So instead of making it null, when we split a new leaf we point the end of new leaf to end pointer of old leaf, and then change the end pointer of old leaf to the newly created leaf.

3) Initially, the Index nodes split was facing a [IndexInsertRecException](http://www.eecs.yorku.ca/course_archive/2013-14/W/4411/proj/javadoc/btree/IndexInsertRecException.html). This was because the index was full and the split had not happened. We found this was due to explicitly mentioning the node type as Index, instead of letting the current page retrieve the type from get\_keyType().

4) Deletion of records from the leaf that was to be copied created ArrayOutOfBound error. This was found to be due to the use of deleteRecords().

We instead used deleteSortedRecords(), which solved our problem.

5) Java Null Pointer error surfaced when the root page was being created, while setting the pointers to Null. We were trying to point to a Page initialized with pid as -1. We changed the pointers to point to a PageId constructor passing -1.