B+Tree Project Report

Division of Labor:

Team 31

· Sakshi(sx3702@mavs.uta.edu, 1001993702) (implementation of insert, omega run)
· Spoorthi Seri(sxs2684@mavs.uta.edu,1002032680) (implementation of delete)
· Neha Thokala(nxt0631@mavs.uta.edu, 1002030631) (documentation)

Overall Status

- Implemented insert and delete functions in BTree class
- Test the changes in Eclipse IDE
- Modified file paths in the make file
- Added log4j JAR dependency to classpath
- After testing locally, copy source files to the UTA omega server

Login at omega and create a directory
-mkdir btproject

After making btproject folder in omega, copy files by running this command in local terminal
-pscp -r /Users/sakshisrivastava/Desktop/uta/Sem3Spring/DBMS/btproject1/btproject1code/ sx3702@omega.uta.edu:/home/s/sx/sx3702/btproject/

SSH to omega and verify
-cd btproject/src/btree/
-make all # compiles the code
-cd ../tests
-make bttest # run test cases

Overview of the Insertion Algorithm:

- The insert method inserts a new key and its associated record ID (RID) into the B+ tree. It first checks if the tree is empty by examining the root page's ID. If the root is empty, a new leaf page is created and the key and RID are inserted into it. If the root is not empty, the _insert method is called with the root page's ID.
- The insertKey method examines the type of the page indicated by presentPageId (an index page or a leaf page) and calls the appropriate method to insert the key and RID into the page. If a split occurs during the insertion, a new page is created to hold the "overflow" records, and a key point to the new page is returned to be inserted into the parent page.
- The insertInIndex method inserts the key and RID into an index page by recursively calling _insert with the appropriate child page ID.
- The insertInLeaf method inserts the key and RID into a leaf page. If there is enough space in the page for the new key and RID, they are simply inserted. If there is not enough space, the page is split into two pages, and the new key and RID are inserted into the appropriate page. The method returns a key to be inserted into the parent index page if a split occurred.

Explanation (process of splitting a full index page into two pages):

A traversal alternate (Alt2) is chosen while implementing the insertion module i.e Alt2) the right pointer needs to be traversed if the key1 value is greater than key2; else traverse the left pointer.

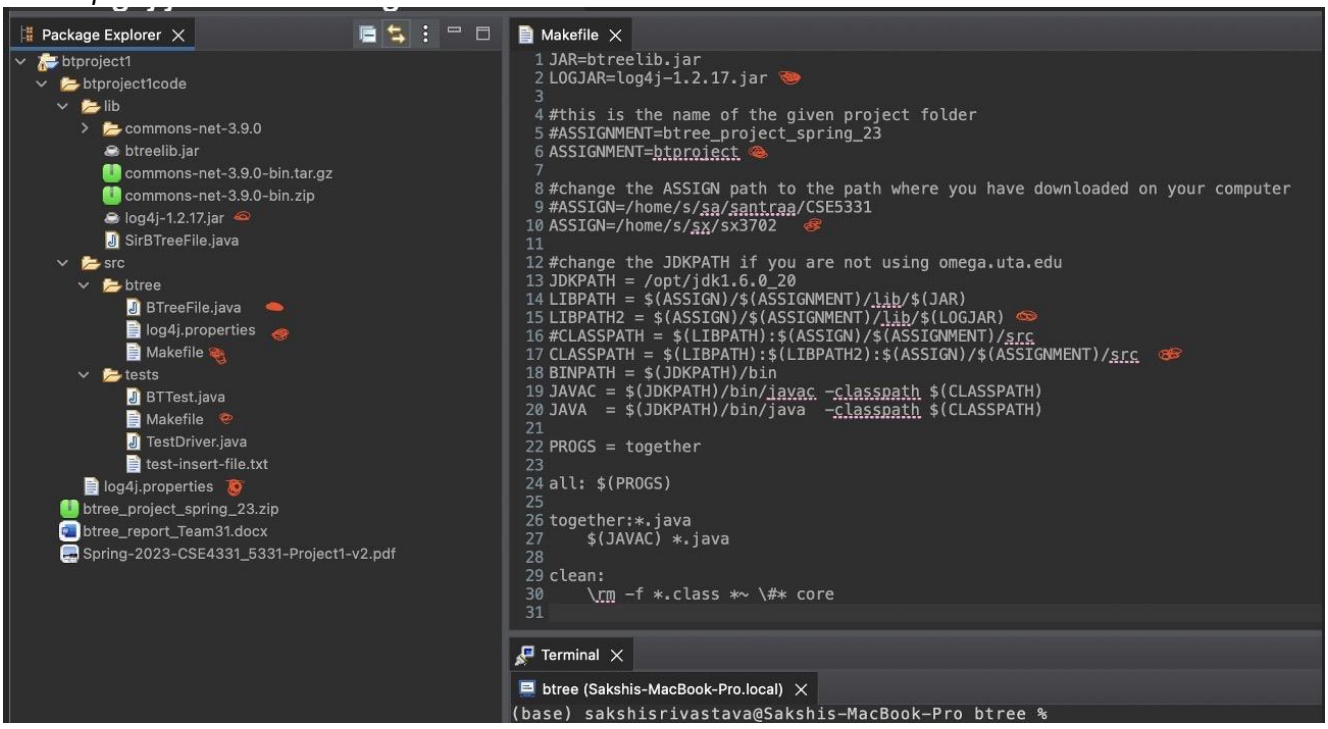
- In a B+ tree insertion, the process starts at the root node and recursively descends the tree until it reaches the leaf node where the new record can be inserted.
- To determine which child node to traverse, we first compare the key of the KeyDataEntry object shiftdataup with the key of the current KeyDataEntry object tempd from the new split index page. If shiftdataup's key is greater than tempd's key, it means that shiftdataup needs to be inserted into the new split index page, which means, creating a new IndexData object using the shiftdataup's data and inserting the key and the new IndexData object into the new split index page using the insertKey() method.
- If shiftdataup's key is less than or equal to tempd's key, it means that shiftdataup needs to be inserted into the present index page.which means, creating a new IndexData object using the shiftdataup's data and inserting the key and the new IndexData object into the present index page using the insertKey() method.
- If the selected child node is an index node, the process is repeated until a leaf node is reached. If the leaf node has enough space to accommodate the new record, it is inserted there. Otherwise, the leaf node is split and the middle key value is propagated up to the parent index node. If the parent index node is full, it is split in turn, and the process is repeated recursively until the root node is reached. If the root node is full, it is split and a new root node is created.

Overview of Delete Algorithm:

- The NaiveDelete method implements a simple deletion algorithm that does not take into account the balance of the tree or the possibility of underflow or overflow. As a result, the method may lead to a poorly balanced or inefficient tree structure
- NaiveDelete is a method in the B+ tree implementation that deletes a key-value pair from the tree using a naive deletion algorithm. The method takes a key and rid it as input, representing the key and the record ID of the key-value pair to be deleted. The method first finds the leaf page where the key is located by calling the findRunStart method. It then searches for the key-value pair in the leaf page and deletes it. The deletion process may involve redistributing records within the leaf page or merging adjacent pages if necessary. Once the deletion is completed, the method returns true if a record was deleted, or false if the record was not found

File descriptions

No new code files were added, however we did add a log4j JAR library as a dependency. This library helps debug info and error logs. We updated the classpath to ensure this library is loaded at compile time



Future Improvements

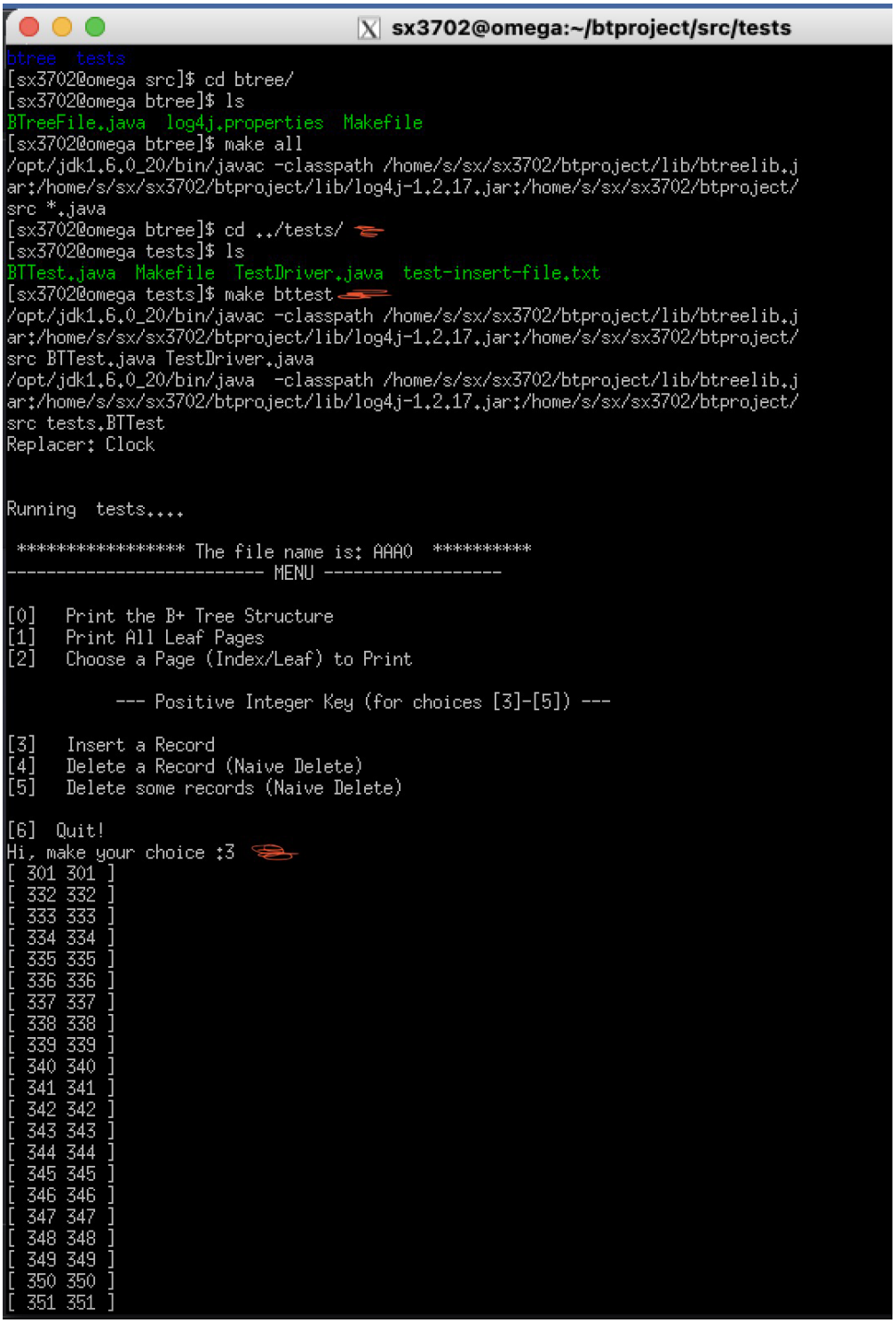
- More advanced deletion algorithms, such as the "redistribution and merging" or "borrowing and redistribution" techniques, can be used to ensure that the tree remains balanced and efficient even after deletions.
- if keys are not properly sorted within a node during insertion or deletion, it may result in a search operation not being able to locate the expected key or returning the wrong key. This can be particularly challenging to identify because the behavior may only be apparent when searching for specific keys or under certain conditions.
- We have to ensure that keys are always sorted within a node and perform sanity checks during insertion and deletion to confirm that the B-tree structure remains valid.
- In the future we can implement an automated testing suite that checks the correctness of the B-tree structure and verifies that search operations return the expected results.
- handle edge cases where keys may be equal or have the same value, as these can also affect the ordering of keys within a node. For example, if multiple keys in a node have the same value, it may be necessary to use a secondary sort key to ensure that the ordering is deterministic.
- It is important to ensure that the B-tree is balanced properly during insertion and deletion operations. This can be achieved by using techniques such as node splitting, node merging, and re-balancing the tree as needed. In the future, it may be helpful to use heuristics or algorithms to determine when the tree needs to be re-balanced, rather than relying solely on manual adjustments or predetermined rules.
- Another related issue is the management of free space within the B-tree file. If nodes are frequently split and merged, it can lead to the fragmentation of the file and a reduction in overall performance. To mitigate this issue, it is important to properly manage the allocation and de-allocation of space within the file and to perform garbage collection operations to consolidate free space and minimize fragmentation.

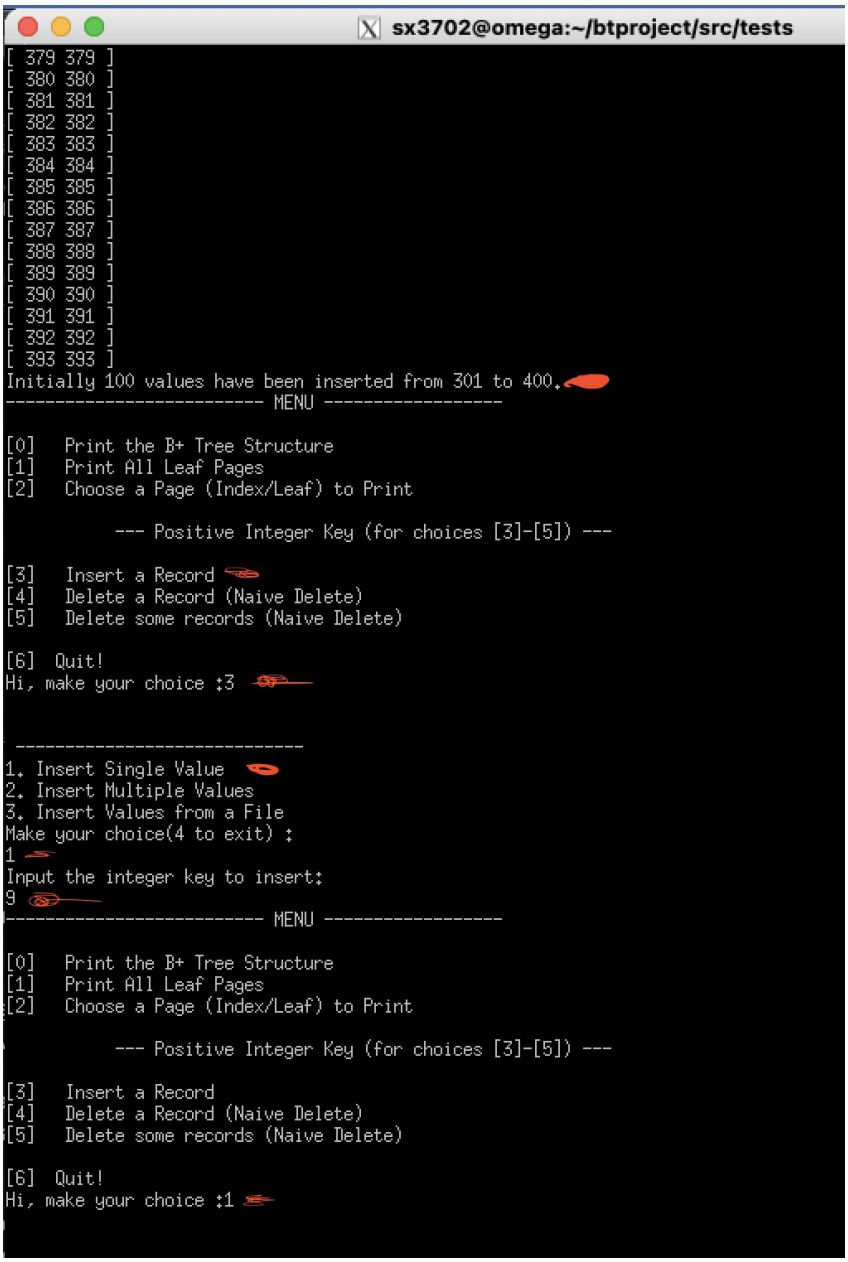
Logical errors handled:

- Null Pointer Exception: The code assumes that certain objects, such as the header page, are not null. If these objects are null, a null pointer exception will be thrown.
- Infinite Loop: If the lo_key passed to the findRunStart method is greater than all the keys in the tree, the loop that searches for the correct leaf page will never exit.
- Unnecessary Object Instantiation: The code instantiates new objects (e.g., BTSortedPage, BTIndexPage, RID) on each iteration of the loop in the findRunStart method, which could be inefficient and unnecessary.
- Incorrect Page Unpinning: In the NaiveDelete method, the code unpins a page before deleting an entry from it, which could result in a PAGE_NOT_PINNED error if the page is subsequently accessed.
- Establishing a Connection with omega on mac and changes made in Makefile is not clearly mentioned.

- The code is refactored to better handle null objects and to reduce unnecessary object instantiation. Additionally, the logic in the findRunStart method is modified to ensure that the loop eventually exits, and the NaiveDelete method is modified to ensure that pages are not unpinned prematurely. unpinPage(splitleaf_Id, true);

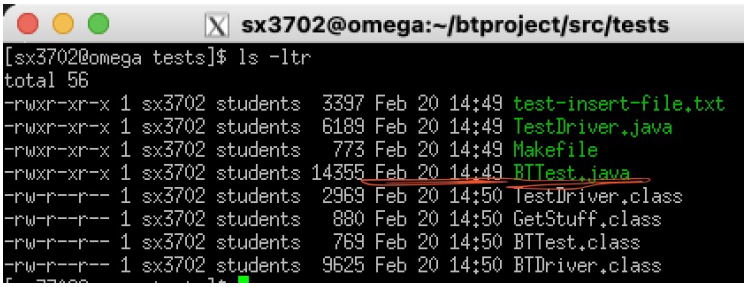
Test on omega Screenshots:




sx3702@omega:~/btproject/src/tests
[379 379]
[380 380]
[381 381]
[382 382]
[383 383]
[384 384]
[385 385]
[386 386]
[387 387]
[388 388]
[389 389]
[390 390]
[391 391]
[392 392]
[393 393]
Initially 100 values have been inserted from 301 to 400.
------------------------- MENU -----------------
[0] Print the B+ Tree Structure
[1] Print All Leaf Pages
[2] Choose a Page (Index/Leaf) to Print
--- Positive Integer Key (for choices [3]-[5]) ---
[3] Insert a Record
[4] Delete a Record (Naive Delete)
[5] Delete some records (Naive Delete)
[6] Quit!
Hi, make your choice :3
1. Insert Single Value
2. Insert Multiple Values
3. Insert Values from a File
Make your choice(4 to exit) :
1
Input the integer key to insert:
9
------------------------- MENU -----------------
[0] Print the B+ Tree Structure
[1] Print All Leaf Pages
[2] Choose a Page (Index/Leaf) to Print
--- Positive Integer Key (for choices [3]-[5]) ---
[3] Insert a Record
[4] Delete a Record (Naive Delete)
[5] Delete some records (Naive Delete)
[6] Quit!
Hi, make your choice :1

```
[6]  Quit!
Hi, make your choice :1


---------------The B+ Tree Leaf Pages---------------

***************To Print an Leaf Page ********
Current Page ID: 3
Left Link      : -1
Right Link     : 4
0 (key, [pageNo, slotNo]):   (9,  [ 9 9 ] )
1 (key, [pageNo, slotNo]):   (301,  [ 301 301 ] )
2 (key, [pageNo, slotNo]):   (302,  [ 302 302 ] )
3 (key, [pageNo, slotNo]):   (303,  [ 303 303 ] )
4 (key, [pageNo, slotNo]):   (304,  [ 304 304 ] )
5 (key, [pageNo, slotNo]):   (305,  [ 305 305 ] )
6 (key, [pageNo, slotNo]):   (306,  [ 306 306 ] )
7 (key, [pageNo, slotNo]):   (307,  [ 307 307 ] )
8 (key, [pageNo, slotNo]):   (308,  [ 308 308 ] )
```

```
-------------------------- MENU ------------------

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

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

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

[6]  Quit!
Hi, make your choice :4
Input the integer key to delete:
301
-------------------------- MENU ------------------

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

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

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

[6]  Quit!
Hi, make your choice :1


---------------The B+ Tree Leaf Pages---------------

***************To Print an Leaf Page ********
Current Page ID: 3
Left Link      : -1
Right Link     : 4
0 (key, [pageNo, slotNo]):   (9,  [ 9 9 ] )
1 (key, [pageNo, slotNo]):   (302,  [ 302 302 ] )
2 (key, [pageNo, slotNo]):   (303,  [ 303 303 ] )
3 (key, [pageNo, slotNo]):   (304,  [ 304 304 ] )
```


sx3702@omega:~/btproject/src/tests
[sx3702@omega tests]$ ls -ltr
total 56
-rwxr-xr-x 1 sx3702 students 3397 Feb 20 14:49 test-insert-file.txt
-rwxr-xr-x 1 sx3702 students 6189 Feb 20 14:49 TestDriver.java
-rwxr-xr-x 1 sx3702 students   773 Feb 20 14:49 Makefile
-rwxr-xr-x 1 sx3702 students 14355 Feb 20 14:49 BTTest.java
-rw-r--r-- 1 sx3702 students 2969 Feb 20 14:50 TestDriver.class
-rw-r--r-- 1 sx3702 students   880 Feb 20 14:50 GetStuff.class
-rw-r--r-- 1 sx3702 students   769 Feb 20 14:50 BTTest.class
-rw-r--r-- 1 sx3702 students 9625 Feb 20 14:50 BTDriver.class