Group-27 Design Report

NAME: Huanran Li CS_LOGINS: huanran WISC_ID: 9075285727
NAME: Tinghe Zhang CS_LOGINS: tinghe WISC_ID: 9075197393
NAME: Ruolei Zeng CS_LOGINS: ruolei WISC_ID: 9078294627

Q1: How often do you keep pages pinned? How efficient is your implementation?

In our code, there is two main part that need the pages pinned.

First, in the insert function, the function itself is called recursively to find the correct node before insertion. We unpinned the pages right after the iteration calls. And the maximum number of pages that is pinned at a time should be the height of the B+ tree we build.

Second, in the scan function, the pages are unpinned right after this node is traversed in the process, which means that only one pages should be pinned during the whole scan process.

As for the efficiency of the program, we believe that we did our best on recursion method, but it should not be the most efficient solution out there because of the complexity of recursion. Other algorithms related to divide and conquer would do a better job on finding the position.

Q2: The additional design choice.

We choose the recursion to be the main algorithm for find the correct spot to insert an element simply because it is cleaner to implement and easier to debug than other existing algorithms.

During the scan process, when we reached the bottom of the B+ tree, we are using sibling pointers to scan from left to right. This would avoid any unnecessary vertical traversal of the tree.

We defined root of the B+ tree to be a leaf node during the first-time initialization. This is because if there are some files that contain small amount of data, which cannot fill up the root node, being a leaf node could make the root itself a container and save one I/O cost and some space.

As for Test design, we have 5 additional tests:

Test4 tests norm splitting in non-leaf nodes.

Test5 tests the case where tree is empty.

Test6 tests the case where there is only one leaf and one node.

Test7 tests the case where there is negative int values.

Test8 tests for consecutive index tests with different order.

Q3: How your implementation would change if you were to allow duplicate keys in the B+ Tree.

If there are duplicate keys, the main structure of the program could still be maintained. In addition, we could implement a hash-array based structure to store all of the duplicate keys into the same position. That is, change the rid-key to a rid-keysPointer, which could point to a set of elements that has the same key value. This would allow duplicate keys while no complicated process was added to the program.