Programming Project 3: B+ Tree Index Manager

Design Decisions Document

Marvin Tan, Nicole Zhang, Zach Brantmeier

Tree Structure

btree.h was modified to add new fields to the NonLeafNodeInt and LeafNodeInt structs. An integer length property was added which keeps track of the number of elements that have been assigned within the keyArray and ridArray fields on both nodes. To go along with this, the constants INTARAYLEAFSIZE and INTARRAYNONLEAFSIZE were updated to account for the modified change in the byte size of the structs.

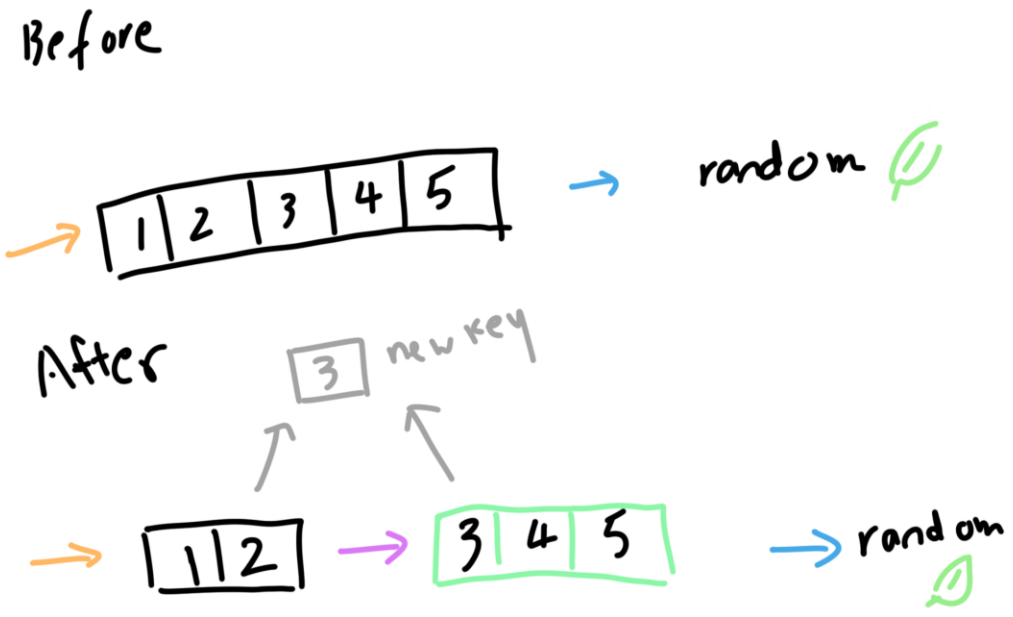
Constructor/Destructor

For the constructor method, we used the BlobFile constructor to create a new file object with the desired index file name. With create_new parameter set to true, if the desired index file has already existed in the file system, the method will throw a FileExistsException, otherwise a new index file will be created. For the file existing case, we will use BlobFile constructor again but set create_new parameter to false to open the desired index file. If a new index file is created, we allocate new header page and root page of the index file to buffer to complete initialization process. If file already exist, we just read those information from the file. After that we unpin all pages used in constructor. Destructor method just followed the pdf instruction (page 4).

Insertion

Node splitting behavior

As leaf nodes only have a pointer to their right node, we decided that leaf nodes which fill up will split into a new node to the right so as to not lose the pointer relationship between the existing node and the one to its left. This way the existing node's right pointer is given to the newly created leaf node, and it is replaced by the pointer to the new node itself.



An illustration of leaf splitting. Non-leaf is very similar, just without all the sibling pointer stuff and the pushed up key is not kept

Rebalancing of parent nodes

Because rebalancing changes to the tree have to propagate upwards from the leaf nodes to their parents, we ultimately implemented a recursive solution to key insertion which is able to perform these changes. We initially intended for a pointer to the parent node to be stored within each of the node structs in the tree, but decided against it because it would increase the storage requirements for the data and was not something that was included in the conventional structure of a B+ tree.

This method of insertion was implemented using additional helper methods described in btree.h which handle the insertion to, and splitting of, both leaf and non-leaf nodes.

Scanning

Right now scanning isimplemented only for the integer type, so there is a check in startScan() that filters out other types of key. A recursive method is used to traverse down the key until it reaches the level above the leaf, and a separate method is called to scan the leaf. The scan stores appropriate variables as specified in the header file for scanNext to use.

ScanNext will first check if a scan is running or not, and it will load the current scanning page if it is not already loaded. In short, page is only pin/un-pined when there is a change in scanned page by traversing through the right sib pointer. If the last value resides in a valid page, the page will not be un-pined until end scan is called so one could choose to end the scan early.