Ryan Horoff
UFID: 8178-1529
UF Email Account: rhoroff@ufl.edu

B+ Tree Implementation Report

My B + Tree is a implemented via two node classes, internal nodes (*InternalNode.java)* and leaf nodes(*LeafNode.java*), both extending an abstract class containing common member variables and functions to be implemented by the specific classes(*BPlusTreeNode.java)*. A container class (*BPlusTreeContainer.java*) holds a root BPlusTreeNode. The *bplustree.java* class is a driver that reads input in from a file, parses it using regex, and performs operations on the root that propagate down depending on what type of node the root is.

BPlusTreeContainer and Operations

The BPlusTreeContainer only contains one node, a root node that can be either an internal node or a leaf node, depending on the current status of the try. The operations and parameters available in this class are as follows.

- Member Variables
    - *root* – a pointer to the root node of this B+ tree, all operations of the b plus tree are performed on the root; it acts as an entry point to the content of the tree
    - *degree* – an int value referring to the degree of the tree
- Functions
    - *public BPlusTreeContainer initialize(int m)*- creates a B+ Tree container with degree m and an initial empty leaf node
    - *public void insert(int key, double value)* – calls the insert function of the root node
    - *public void search(int key)* – calls the search function of the root node
    - *public void rangeSearch(int begin, int end)* – starts a range search by search root node for the *begin* key and traversing the last level linked list until the end key is encountered, or the *end* key's range is reached
    - *public void delete(int keyToDelete)* – calls the root nodes delete function

The container itself is simply an entry point to the tree; most of the structure is defined between the relationships between the nodes

BPlusTreeNode, Operations and Extended Classes

The BPlusTreeNode class contains common elements and functions between both LeafNodes and InternalNodes. This allows all containers of list of nodes to be accessed in traversals for insertions and deletions, as well as allowing for much cleaner and simpler code when it comes to inserting and deleting.

- Member Variables

    - *InternaNode Parent* – a pointer to the parent of the current node, by default set to null
    - *ArrayList<Integers> Keys* – a list of keys that the current node contains; every node contains keys, regardless of whether or not it is a leaf node or internal node
    - *static int MAXNODESIZE* – the current degree of the tree, and the max size that the node can be before a split must occur
- Functions
    - *abstract public BPlusTreeNode split()* – an abstract split function to fix overfull nodes
    - *abstract public BPlusTreeNode insert(int key, int double)* – an abstract insert function to populate nodes

Both leaf nodes and internal nodes have the potential to become over/underfull, so the abstract functions allow for splitting and merging to be done on any type of node without writing specialized code for each combination of internal nodes to one leaf node.

Leaf Nodes

Leaf nodes contain the actual Key Value dictionary pairs that are to be stored in the B+ Tree. As complex data structures were not allowed, the node stores both the keys and values in separate ArrayLists. When it comes time to sort the values to keep the tree consistent and allow for insertions and deletions, the Keys are sorted using a simple bubble sort, and the Values are swapped based off of the indices that are swapped during the sorting of the Keys. In order for the tree to stay correct, the keys and values of the nodes must at all times remain sorted. Leaf nodes lack children, as they are leaves in the tree. They do, however, contain pointers to their left and right leaf node siblings, so as to form the doubly linked list that is indicative of a B+ Tree.

- Member Variables
    - *InternaNode Parent* – a pointer to the parent of the current node, by default set to null
    - *ArrayList<Integer> Keys* – a list of keys that the current node contains; every node contains keys, regardless of whether or not it is a leaf node or internal node
    - *ArrayList<Integer>Values* – a list of the values associated with the keys, matched index by index with the Keys ArrayList.
    - *static int MAXNODESIZE* – the current degree of the tree, and the max size that the node can be before a split must occur
    - *public LeafNode leftSibling* – the left sibling in the leaf level doubly linked list
    - *public LeafNode rightSibling* – the right sibling in the leaf level doubly linked list
- Functions
    - *LeafNode(int m)* – creates an empty leaf node with degree of MAXNODESIZE, inherited from the abstract parent class
    - *public BPlusTreeNode insert(int key, double value)* – Inserts the key value pair directly into the leaf nodes Key ArrayList and Values ArrayList respectively, returning the node that will replace this node in the tree.
        - If the insertion causes this node to be overfull, this method will call the split method and return the node that results from the split function call, which is explained in more detail below.
        - If the insertion does not cause this node to be overfull, then the Keys and Values are sorted in one Bubble Sort and returned

    - *public BPlusTreeNode split()* – splits an overfull node and returns an internal node and the children leaf nodes resulting from a split
        - This method first checks whether or not the node to be split has a parent. If it does, then the function will perform all of the necessary insertions on *all* of the children of the current node (*this.Parent.children*), allowing for the internal nodes internal ArrayLists to be sorted so as to maintain the BPlusTree. A new sibling is added for the current leaf node, and all of the values below the split index (the middle most value of the node) are added to this sibling, which is then added as a separate child of the parent
        - If there is not already a parent node (this only happens if the root is a leaf node), then the above happens, but an internal node is created to be used as the parent for the split. This parent node is then returned and used to replace the current leaf node.
        - All member lists of leaf node are always sorted and all children of the internal node that is created or used by the leaf node split are always sorted
    - *public BPlusTreeNode merge()* – used when a leaf node must be merged due to a deficiency

Internal Nodes

Internal nodes also contain keys, but instead of values maintain a list of the children of the current node for traversal of the B+ Tree. All InternalNodes have a Boolean isLeaf variable that is always set to false, so that during traversal the node can be quickly identified without having to look further into. The internal nodes can also be split, resulting in two internal nodes.

- Member Variables
    - *InternaNode Parent* – a pointer to the parent of the current node, by default set to null
    - *ArrayList<Integer> Keys* – a list of keys that the current node contains; every node contains keys, regardless of whether or not it is a leaf node or internal node
    - *ArrayList<BPlusTreeNode>children* – a list of the children of the current node, which can be both more internal nodes or leaf nodes
    - *static int MAXNODESIZE* – the current degree of the tree, and the max size that the node can be before a split must occur
- Functions
    - *InternalNode(int m)* – creates an empty internal node with degree of MAXNODESIZE, inherited from the abstract parent class, and an empty list of children
    - *public BPlusTreeNode insert(int key, double value)* – Inserts the key value pair into the child of the internal node that falls between a certain range
        - If the insertion causes this node to be overfull, this method will call the split method and return the node that results from the split function call, which is explained in more detail below.
        - If the insertion does not cause this node to be overfull, then the Keys and Children are sorted in one Bubble Sort and this node is returned
    - *public BPlusTreeNode split()* – splits an overfull node and returns an internal node and the children internal nodes resulting from a split
        - In a similar manner to the leaf node, this method first checks whether or not the node to be split has a parent. If it does, then the function will perform all of the

    necessary insertions on *all* of the children of the current node (*this.Parent.children*), allowing for the internal nodes internal ArrayLists to be sorted so as to maintain the BPlusTree. A new sibling is added for the current internal node, and all of the values below the split index (the middle most value of the node) are added to this sibling, which is then added as a separate child of the parent
    - If there is not already a parent node, then the split will use the parent node to perform the other split operations.
- *public BPlusTreeNode merge()* – used when a internal node must be merged due to a deficiency