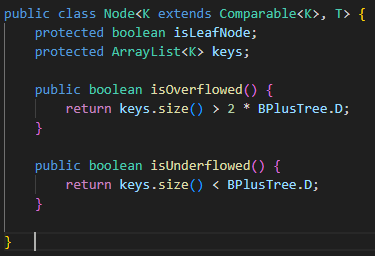
**B Plus Tree Project**

**Design:**

During the design phase of the project, we configured seven java class implementation files. The seven classes listed are BPlusMain.java, BPlusTree.java, IndexNode.java, KeyValuePair.java, LeafNode.java, Node.java and Utils.java.

In the Initial stage, we have the BPlusMain.java file running at startup, which generates a console prompt to provide input . Now we start by loading partfile.txt from the directory and then inserting the key value pair of each element into a LeafNode that uses the Node class, we do a bulk insert to load the entire file into Bplustree.

The BPlusTree.java file contains a class called BPlusTree that holds all the functionality of what we need to use for this project. It also contains a Node class with two properties in the image below. The key variable is an arrayList which will be a data structure holding keys and blocks up to 16. The isLeaf variable will determine if the node is a leaf or not.

Once all the elements are loaded, the user is then presented with selection commands, such as Load BPlus File, Search an Element, Print BPlus Tree, Insert an element, Delete an element, Update an element , exit When the Insert or delete is done First select their Key Do - Value will be inserted. If a key exists, it cannot be inserted, but it can be deleted. Otherwise, if Key is not present, an insert can be performed, but a delete cannot be performed. When inserting, an element Key-value is provided as input to the prompt, which sets the best leaf node position and sorts the input.

A black screen with white text

Description automatically generated

A screenshot of a computer program

Description automatically generatedA screenshot of a computer

Description automatically generated

Search functionality will be taking a key value as input value and then it starts searching all the nodes iteratively, until it finds a match. It will do a binary search for every iteration, so to improve the optimization.

A black screen with white text

Description automatically generated

Display functionality will initially loadup all the leafnodes into a array and then gives options to display the next 10 elements or to exit the display functionality It will keep a record of all the leafnode values.

A screenshot of a computer

Description automatically generated  
  
Update functionality will take the key value to search the value which needs to be updated and then it prompts to enter the new value will needs to be updated, providing the input. The value will be stored in a array, which can be stored on exit the program.

A screenshot of a computer program

Description automatically generated

When you exit the program, you are given the option to save the current file description in partfile.txt or not. Regardless of which method you choose, you will be given the final result of the tree’s incisions, fusions, and depth.

A screenshot of a computer program

Description automatically generated

**Algorithms:**

The two algorithms that were mainly used for the B plus tree are insert and delete. The insert and delete algorithms are shown below in pseudo code.

public class Node<K extends Comparable<K>, T> {

protected boolean isLeafNode;

protected ArrayList<K> keys;

public boolean isOverflowed() {

return keys.size() > 2 \* BPlusTree.D;

}

public boolean isUnderflowed() {

return keys.size() < BPlusTree.D;

}

}

**Insert**

insert(string key, string data) {

If (root is NULL) add key and data in root

return

while (current is not a leaf) If (key < current.key) go to pointer before current.key

If (current.size < max node size)

Add key and data in node

else

split node into two insert key and data into the correct place if (parent.size < max index node size) add first key from new node to parent add pointer to the new node

else while (parent.size > max index node size) split parent insert first key from new index node to parent node add pointer to the new node current = parent parent = find the parent of the parent variable

}

public void insert(K key, T value) {

Entry<K,Node<K,T>> entry = new AbstractMap.SimpleEntry<K,Node<K,T>>(key, new LeafNode<K, T>(key,value));

if(root == null)

root = entry.getValue();

Entry<K,Node<K,T>> newChildEntry = insertHelper(root,entry,null);

if (newChildEntry == null)

return;

else {

IndexNode newRoot = new IndexNode(newChildEntry.getKey(),root,newChildEntry.getValue());

root = newRoot;

return;

}

}

**Remove**

remove(string key, string data) {

If (root is NULL) return

while (current is not a leaf) If (key < current.key) go to pointer before current.key

If (current.size > (max node size / 2)) remove key and data in node

else join left or right node into current node remove key and data from the node if (parent.size > (max index node size / 2)) remove pointer from parent pointing to the old node

else while (parent.size <= (max index node size / 2)) join left or right node into current node remove key and data from new index node to parent node remove pointer to the old node current = parent parent = find the parent of the parent variable

}

public void delete(K key) {

if (root == null)

return;

LeafNode leafFound = (LeafNode)treeSearch(root, key);

if (leafFound == null)

return;

Entry<K,Node<K,T>> entry = new AbstractMap.SimpleEntry<K,Node<K,T>>(key,leafFound);

Entry<K,Node<K,T>> oldChildEntry = deleteHelper(root,root,entry,null);

if (oldChildEntry == null) {

if (root.keys.size() == 0){

if (!root.isLeafNode)

root = (Node)((IndexNode)root).children.get(0);

}

return;

} else {

int i = 0;

while (i < root.keys.size()){

if (oldChildEntry.getKey().compareTo(root.keys.get(i)) == 0)

break;

i++;

}

if (i == root.keys.size())

return;

root.keys.remove(i);

((IndexNode)root).children.remove(i+1);

return;

}

}

**Challenges:**

The most challenging parts of the project were implementing the insert and delete function. And inserts and deletions take a little more time to make sure we don’t want to override any tree properties. For example, insert increases the node size by one, and we need to keep the node size at a certain size, so we had to think about it

How is it classified? Same with delete, except in that case, we needed a minimum size for the node, so we had to figure out how to add it to someone else.

Another issue we encountered was that we were first looking at B tree descriptions instead of B plus tree descriptions. The references included the insert function of the key, but no data. So, we had to figure out how to do it ourselves.

One final issue is to compare the strings with other strings. In a B+ tree, numbers are often used internally to indicate where the ID should go in the tree. Normally a comparison would be made between these two numbers, but in this case we were using them as string IDs (i.e. ABC-001). So, we had to find a way to do that.

**Conclusion:**

In conclusion, our B+ tree project was a valuable learning experience. we successfully completed the project with all the required functionalities. But this experience taught us a lot about the architecture and effectiveness of the B+ trees, and the challenges associated with implementing them on a windows Platform We learned valuable lessons from this experience and are determined not to same mistake again in future.

Throughout the project, we gained a solid understanding of how a B+ tree works and how it is built in a programming language like Java. We encountered various obstacles during the implementation of the algorithms, especially for insert and delete functions. We overcame these challenges by consulting textbooks and external resources.

Overall, we found that building a B+ tree was an effective way to gain hands-on experience in building a list style tree. We are proud of the knowledge gained from this project and will use this knowledge to improve our programming skills in the future

**References:**

<https://www.geeksforgeeks.org/insertion-in-a-b-tree/><https://www.programiz.com/dsa/b-plus-tree><https://github.com/linli2016/BPlusTree>

By :

SHASHI KUMAR K (1338369)

DHANA KRISHNA P (1325505)

NIDHISHA REDDY B (1324518)

KETHANA KATEPALLI (1321620)