# COP 5536: Advance Data Structures

# Implementation of B+ Tree


| | |
|---|---|
| Name: | Hemanth Kumar Malyala |
| UFID: | 6348-5914 |
| Mail Id: | hmalyala@ufl.edu |

# Project Description

This project is about implementation of B+ trees in Java. B+ tree is a data structure used for storing Key, Value pairs. The attribute of B+ tree which differentiates it from other data structures is the same level of its leaf nodes. All the elements are always present in the leaf node. B+ Tree always have all the external nodes i.e. leaf nodes at same level. So, It's a self-balancing data structure with complexity of log(x) time. During insertion it checks the order of B+ tree. If the total number of inserted key value pair in node reach to the order value, then it will split the node and distribute the data equally. The data stored in a B+ Tree is in the sorted order.

This assignment is to implement the four functionalities of B+ Tree:

1. **Initialization:** The first & foremost step was to initialize a tree of order 'm' of the B+ tree.
2. **Insert Key Value Pair:** Insertion is done by calling insert function and passing the key and value as parameter. It inserts the key at its correct sorted position. So that we get a properly sorted tree. The inserted value then goes to the leaf node and final get its position there.
3. **Delete:** a function to delete the value of a key-value pair.
4. **Search:** it will search a key in the B+ Tree in log(n) time and then return the value associated to that key.
5. **Search Range:** it will search the range of keys i.e. Key1<=keys<=Key2 in the B+ tree. Then it will print the values associated with that key range.

## Structure of the Program

In this code, B+ Tree is implemented through 4 classes.

1. BPlus_Tree
2. BPlus_Tree_Node
3. BPlus_Tree_Inner_Node
4. BPlus_Tree_Leaf_Node
5. bplustree

Two of the above classes describe the data structure used to implement the B+ tree. They contain constructors, pointers for traversing across the tree and member functions to perform the insert, delete and search operation. BPlus_Tree_Node determines the structure of B+ tree as it grows, whereas BPlus_Tree contains the basic methods of insert, delete and search which are implemented rigorously in the other two classes.One of these two classes is an abstract class which contains all the methods required to build a B+ tree. The other two classes inherit and implement the functions defined this, as the tree grows/shrinks.

Class bplustree instantiates the class “BPlus_Tree”, reads the data from the input_file and redirects the output to another file. The <key,value> pairs to be inserted into the B+ tree are specified in the input_file, and this class reads in the input one line at a time. Similarly the output of each search() member function call is written out to the output_file. BufferedReader and BufferedWriter are used for this purpose. StringTokenizer is used to decode the input_file to identify the type of the operation to be performed (i.e. which method to be called) and also the appropriate arguments.

## FUNCTIONS USED

```
public void insert(TKey key, TValue value)
```

**Description:**

This is the method used to insert values into the B+ Tree.

Note: The data type TKey and TValue are generic i.e. we can store <Key,Value> pairs of any type into the tree.

It's return type is void, which means it only places the <Key,Value> pair into the tree and doesn't return any value.

**Input type:**

The arguments to this method are the <Key,Value> pair which could be of any data type.

```
public void delete(TKey key)
```

**Description:**

This is the method used to delete values from the B+ Tree.

It's return type is void, which means it only removes the <Key,Value> pair specified by the <Key> passed as argument and doesn't return any value.

**Input type:**

The arguments to this method is just the <Key>, Both the key and it's corresponding value is deleted from the tree..

```
public TValue search(TKey key)
```

**Description:**

This is the method used to search the <Key,Value> pair from the B+ Tree.

It's return type is void, which means it only finds the <Key,Value> pair associated with the <Key> passed as argument and doesn't return any value.

**Input type:**

The arguments to this method is just the <Key> to be searched.

```
Private BPlus_Tree_Leaf_Node<TKey, TValue>
 find_leaf_node_has_key(TKey key)
```

**Description:**

This is the method used to find out if the current node is internal or external node.

It's return type is BPlus_Tree_Leaf_Node, i.e it return the leaf node of the argument passed.

**Input type:**

The arguments to this method is just the <Key> to be searched.

```
     public abstract TreeNodeType getNodeType();
```

**Description:**

It is an abstract method which is implemented by the class inheriting it while performing the insert operation.

**Input type:**

This method doesn't take in any argument.

```
public boolean isOverflow() {
```

```
public BPlus_Tree_Node<TKey> exceed()
```

**Description:**

It is an abstract method which is implemented by the class inheriting it while performing the insert operation. They determine whether or not the node into which the <Key,Value> is being inserted is overflowed and the corresponding parent node.

One function returns a Boolean value of true or false while the other one returns the node i.e. overflown

**Input type:**

Both functions do not take in any argument.

```
public abstract BPlus_Tree_Node<TKey> split();
```

```
public abstract BPlus_Tree_Node<TKey> traverse_above(TKey key,
BPlus_Tree_Node<TKey> leftChild, BPlus_Tree_Node<TKey> rightNode);
```

```
public abstract TKey transfer_adjacent(TKey key_1,
 BPlus_Tree_Node<TKey> sibling, int i);
```

```
     public              abstract              BPlus_Tree_Node<TKey>
merge_child(BPlus_Tree_Node<TKey> left_child, BPlus_Tree_Node<TKey>
right_child);
```

```
     public     abstract     void     merge_adjacent(TKey     key_1,
BPlus_Tree_Node<TKey> rightSibling);
```

```
     public    abstract    void    shift_child(BPlus_Tree_Node<TKey>
borrower, BPlus_Tree_Node<TKey> lender, int borrowIndex);
```

**Description:**

All the above abstract functions are overridden in the inheriting class. They come into picture while performing the delete operation on the B+ tree.

Individually they do not perform a meaningful operation, collectively they make a perfect delete operation by making sure that neither the current node nor the adjacent nodes are empty or below the Math.ceil(d/2)-1.

```
public BPlus_Tree_Node<TKey> getChild(int index)

public void setChild(int index, BPlus_Tree_Node<TKey> child)

public TreeNodeType getNodeType()

public int search(TKey key)

private void insert_in(int index, TKey key, BPlus_Tree_Node<TKey>
leftChild,

BPlus_Tree_Node<TKey> rightChild)

public BPlus_Tree_Node<TKey> traverse_above(TKey key,
BPlus_Tree_Node<TKey> leftChild, BPlus_Tree_Node<TKey> rightNode)

private void remove_at(int index)

public BPlus_Tree_Node<TKey> split()

public BPlus_Tree_Node<TKey> merge_child(BPlus_Tree_Node<TKey>
left_child,

BPlus_Tree_Node<TKey> right_child)

public void shift_child(BPlus_Tree_Node<TKey> temp_1,
BPlus_Tree_Node<TKey>
```

```
temp_2, int loc)
```

```
public void merge_adjacent(TKey key_1, BPlus_Tree_Node<TKey>
rightSibling)
```

```
public TKey transfer_adjacent(TKey key_1, BPlus_Tree_Node<TKey>
sibling, int loc)
```

**Description:**

The methods above are those of the abstract class BPlus_Tree_Node which are overridden while performing the insert and remove operations on the leaf and internal nodes of the B+ Tree. The return types and input data types are same as discussed above. However, few of the above methods are overloaded in few cases based on the value returned from its preceding function.

Similar to the methods described earlier, all the methods together hold the structure of the B+ Tree.