Advanced Data Structures
Report B+ trees implementation

Aastha Jhunjhanuwala
UFID: 55271081
UF mail ID: ajhunjhanuwala@ufl.edu

Description:

The aim of this project is to create a B+ tree implementation that performs actions like insert, delete and searches. These trees are used to perform dynamic multilevel indexing and are extremely powerful and efficient. B+ trees store the data nodes only at the leaf nodes of the tree. The remaining internal nodes are simply pointers to the children data nodes.

B+ trees use balancing techniques to ensure that the tree is always a full tree. No internal node has empty children nodes and incase such situation arises the tree self-balances itself. The data structure of the leaf node comprises of key value pairs and a doubly linked list across the siblings. The internal nodes all store only keys of the first key of the right child and the pointers to the children. This makes it easy to traverse the tree and produce results quickly using binary search techniques. This program takes the operations to be performed as in input in an input file placed at the same directory and output the search results in an output file called output_file.txt.

Running the program:

A make file is included in the folder so after unzipping the folder run the following command:
$make

After compilation run the following command to run the code:
$java bplustree <input_file>

This will store the result in:
output_file.txt

Class Structure:

1.Node Class

This class stores the data structure of each node in the B+ tree. Some fields are common to both the internal and leaf nodes but fields that are specifically a leaf node attribute or an internal node attribute are specified in the below fields tab. The flag determines if the node is an internal node (value = 0) or external or leaf node (value=1).

Class Node

java.lang.Object
    Node

```
public class Node
extends java.lang.Object
```

The Class Node contains the data structure of each node of the B+ tree.

Fields:

Field Summary

Fields

Modifier and Type	Field and Description
`java.util.List<Node>`	**`children`** Only internal node attribute.
`int`	**`flag`** flag: 0 = internal node 1 = leaf node
`java.util.List<java.lang.Integer>`	**`keys`** Store the list of keys at the node
`Node`	**`next`** Only leaf node attribute.
`int`	**`nodeType`** Store the type of node
`int`	**`numOfKeys`** Store the number of keys in the node
`Node`	**`parent`** Stores a pointer to the parent node
`Node`	**`previous`** Only leaf node attribute.
`java.util.List<java.lang.Double>`	**`values`** Only leaf node attribute.

Methods:

Constructor Summary

Constructors

Constructor and Description
`Node()` Instantiates a new node of a B+ tree.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`void`	`createInternalNode()` Create a new internal node and set flag.
`void`	`createLeafNode(int key, double value)` Create new leaf node in the B+ tree.

2.Class BPlusImpl:

This class performs the actual implementation of the operations carried out in the B+ tree. There are methods including insert key-value pair, delete nodes, search for values by key and within the key range. This class contains multiple helper functions to enable the above specified main functions for example merge and split nodes, search inside nodes, wtc.

Class BPlusImpl

java.lang.Object
    BPlusImpl

```
public class BPlusImpl
extends java.lang.Object
```

The Class BPlusImpl executes all the B+ tree modification functions like insert, delete, search.

Fields:

Field Summary

Fields

Modifier and Type	Field and Description
`int`	`m` Stores the order of the tree
`Node`	`root` Root node of the tree

Methods:

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
void	**delete**(int key) Search the B+ tree for the key and deletes corresponding key-value pair from the tree.
void	**initialize**(int m)
void	**insert**(int key, double value) Insert a new key value pair in the B+ tree.
void	**insertLeafNode**(int key, double value, **Node** node) Insert the new key value pairs in the respective lead node location
void	**merge**(**Node** existingNode, **Node** newNode) When any new node is to be merged with an exsiting node.
void	**mergeInternalNodes**(**Node** current, int index) If a internal node has an underflow after the delete, it merges the internal node with a sibling.
int	**mergeLeafSibling**(**Node** current, int index) If a leaf node has an underflow after the delete, it merges the leafnode with a sibling.
void	**search**(int key) Search the B+ tree for the key and returns corresponding value from the tree.
void	**search**(int keyStart, int keyEnd) Search the B+ tree for the range of keys taken as input and returns all the values of the in-between keys from the tree.
int	**searchInNode**(**Node** node, int key) Search exact key location in any node: binary search.
void	**splitInternalNode**(**Node** toBeInsertedNode, **Node** childNode, **Node** currentNode, boolean leafChild) Merges the parent of the new split child node into an existing internal node.
void	**splitLeafNode**(**Node** node) If leaf node overflows during insert, this function is called where is splits the leaf node and makes the middle node as parent.

3.Class bplustree:

This class performs the wrapper activity to run the commands from a given input file and print the output in the file output_file.txt

Class bplustree

java.lang.Object
    bplustree

```
public class bplustree
extends java.lang.Object
```

Class bplustree is used to run and print commands of the B+ tree execution in a separate output file.

Method:

Method Summary

All Methods | Static Methods | Concrete Methods

Modifier and Type	Method and Description
static void	main(java.lang.String[] args)

Functional Prototype:

Node class:

public void createLeafNode(int key, double value)

This method creates a new leaf node and inserts the key value pair into the node.

createLeafNode

```
public void createLeafNode(int key,
                           double value)
```

Create new leaf node in the B+ tree.

Parameters:

`key` - the key to be inserted in the new leaf node

`value` - the corresponding value to be inserted at the new leaf node

public void createInternalNode()

This method creates a new internal node and sets the flag.

createInternalNode

```
public void createInternalNode()
```

Create a new internal node and set flag.

BPlusImpl Class:

public void initialize(int m)

This method initializes to set order as m and root as null.

initialize

```
public void initialize(int m)
```

public void insert(int key, double value)

This method inserts a new key value pair in the B+ tree. It is the main insert function called from the input file that then calls multiple helper functions to perform complete end to end insert operation that may include handling overflow at leaf and internal nodes, searching inside nodes to locate the index of a key, etc.

insert

```
public void insert(int key,
                   double value)
```

Insert a new key value pair in the B+ tree.

Parameters:

`key - the key to be inserted`

`value - the value to be inserted`

public void insertLeafNode(int key, double value, Node node)

Insert the new key value pairs in the respective lead node location. This method simply performs an insert in the leaf node and the parent function then checks for overflow.

insertLeafNode

```
public void insertLeafNode(int key,
                           double value,
                           Node node)
```

Insert the new key value pairs in the respective lead node location

Parameters:

`key - the key to be inserted in the leaf node`

`value - the value to be inserted in the leaf node`

`node - the node where the key value pair is to be inserted`

public int searchInNode(Node node, int key)

Search exact key location in any node: binary search. Returns existing key index or the next key index.

searchInNode

```
public int searchInNode(Node node,
                        int key)
```

Search exact key location in any node: binary search. Returns existing key index or the next key index.

Parameters:

`key - the key to be searched in the node`

`node - the node where the key is to be searched`

Returns:

`the index at which the key exists or should exist`

public void splitLeafNode(Node node)

If leaf node overflows during insert, this function is called where is splits the leaf node and makes the middle node as parent. This function first performs the split of lead node by creating a parent node and assigning the children. It then calls the split internal node function indicating that an internal node split might be required.

splitLeafNode

```
public void splitLeafNode(Node node)
```

If leaf node overflows during insert, this function is called where is splits the leaf node and makes the middle node as parent.

Parameters:

```
node - the node that overflows and needs to be split
```

public void splitInternalNode(Node toBeInsertedNode, Node childNode, Node currentNode, boolean leafChild)

Merges the parent of the new split child node into an existing internal node.

If internal node overflows during insert, this function is called where is splits the internal node and makes the middle node as parent. It first performs a check to see if the internal node is overflowing and then performs split recursively on the parent of each internal node till we reach a non-overflow parent, where it stops.

splitInternalNode

```
public void splitInternalNode(Node toBeInsertedNode,
                              Node childNode,
                              Node currentNode,
                              boolean leafChild)
```

Merges the parent of the new split child node into an existing internal node. If internal node overflows during insert, this function is called where is splits the internal node and makes the middle node as parent.

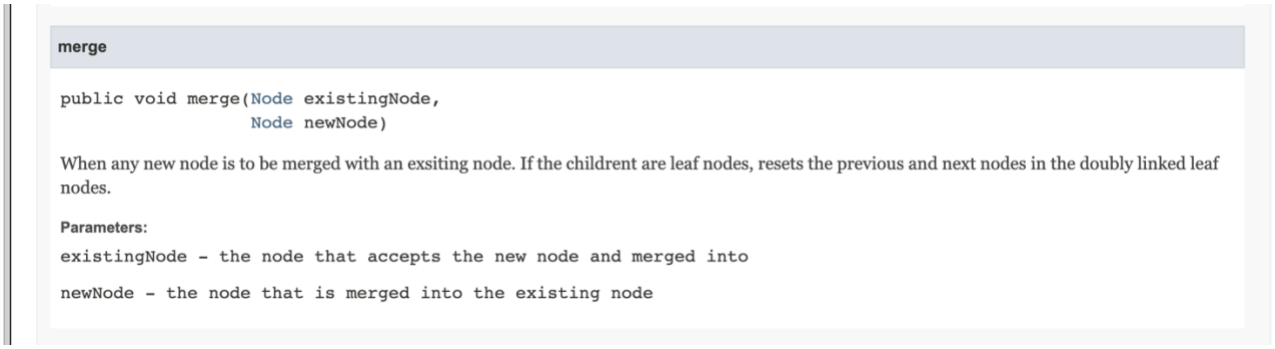
Parameters:

```
toBeInsertedNode - the new node to be inserted, parent of the split child nodes
childNode - the right child of the toBeInsertedNode
currentNode - the existing node where the toBeInsertedNode has to be merged
leafChild - stores true if the immediate child is a leaf node
```

public void merge(Node existingNode, Node newNode)

When any new node is to be merged with an exsiting node.
If the childrent are leaf nodes, resets the previous and next nodes in the doubly linked leaf nodes.
This helper function performs a merge of an internal node caused due to split with an existing

internal node. It checks the children leaf node affected and updates their linked list pointers accordingly along with parents.



merge

```
public void merge(Node existingNode,
                  Node newNode)
```

When any new node is to be merged with an exsiting node. If the childrent are leaf nodes, resets the previous and next nodes in the doubly linked leaf nodes.

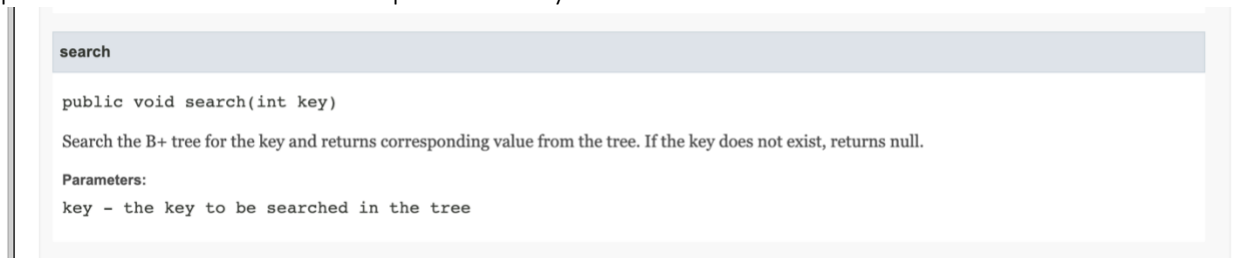
Parameters:

`existingNode - the node that accepts the new node and merged into`

`newNode - the node that is merged into the existing node`

public void search(int key)

Search the B+ tree for the key and returns corresponding value from the tree.
If the key does not exist, returns null. This function traverses the tree and reaches a leaf, then prints the value or null for the particular key.



search

```
public void search(int key)
```

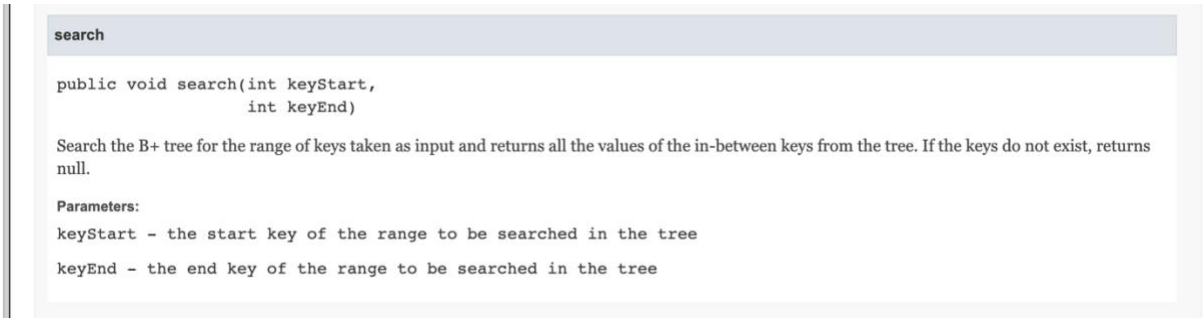
Search the B+ tree for the key and returns corresponding value from the tree. If the key does not exist, returns null.

Parameters:

`key - the key to be searched in the tree`

public void search(int keyStart, int keyEnd)

Search the B+ tree for the range of keys taken as input and returns all the values of the in-between keys from the tree.
If the keys do not exist, returns null.



search

```
public void search(int keyStart,
                   int keyEnd)
```

Search the B+ tree for the range of keys taken as input and returns all the values of the in-between keys from the tree. If the keys do not exist, returns null.

Parameters:

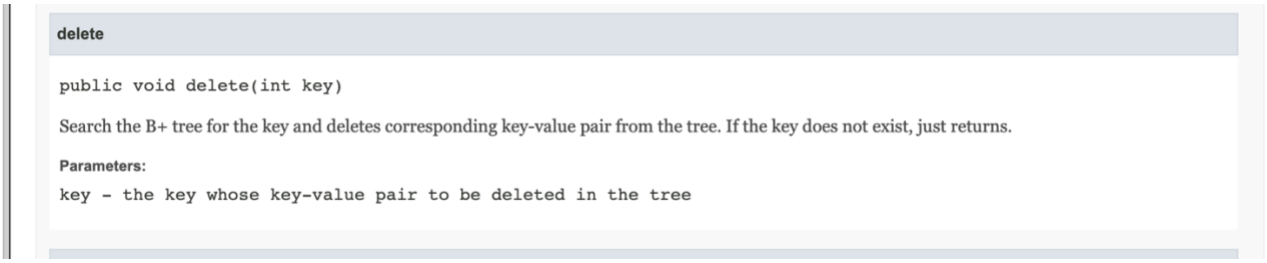
`keyStart - the start key of the range to be searched in the tree`

`keyEnd - the end key of the range to be searched in the tree`

public void delete(int key)

Search the B+ tree for the key and deletes corresponding key-value pair from the tree.

If the key does not exist, just returns. This function uses helper functions like merge internal and leaf nodes to handle underflow at a particular node. If a node becomes deficit after a delete it invokes the merge leaf sibling method.



delete

```
public void delete(int key)
```

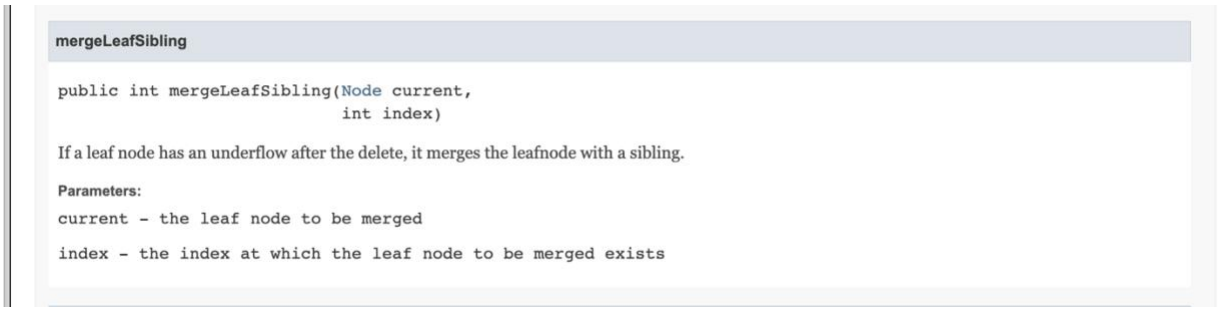
Search the B+ tree for the key and deletes corresponding key-value pair from the tree. If the key does not exist, just returns.

Parameters:

`key - the key whose key-value pair to be deleted in the tree`

public int mergeLeafSibling(Node current, int index)

If a leaf node has an underflow after the delete, it merges the leafnode with a sibling. This method looks for left and right siblings with enough nodes to borrow and if it is not met, it performs a merge using the internal parent node with itself and calls the merge internal nodes to handle this scenario.



mergeLeafSibling

```
public int mergeLeafSibling(Node current,
                            int index)
```

If a leaf node has an underflow after the delete, it merges the leafnode with a sibling.

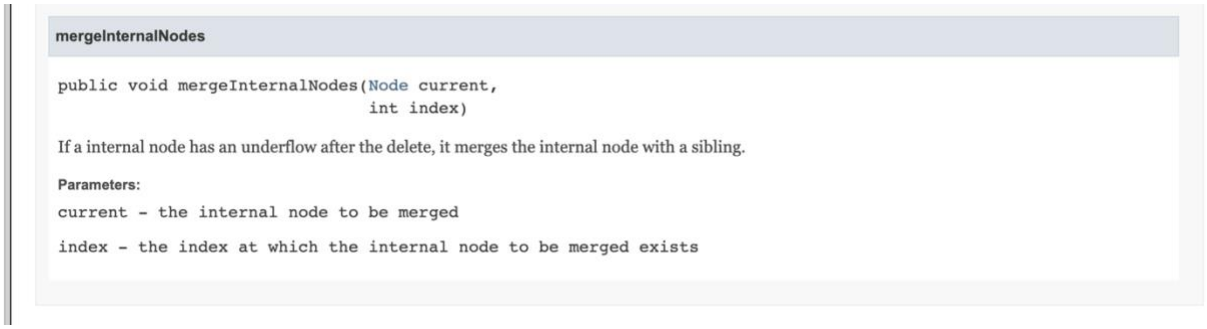
Parameters:

`current - the leaf node to be merged`

`index - the index at which the leaf node to be merged exists`

public void mergeInternalNodes(Node current, int index)

If an internal node has an underflow after the delete, it merges the internal node with a sibling. This method checks if an internal node has been merged with a child node and checks if there is a cascading underflow in this case. It executes the merge function recursively until we reach a parent node with sufficient children nodes.



mergeInternalNodes

```
public void mergeInternalNodes(Node current,
                               int index)
```

If a internal node has an underflow after the delete, it merges the internal node with a sibling.

Parameters:

`current - the internal node to be merged`

`index - the index at which the internal node to be merged exists`