B+ Tree

UFID: 8115 5459
Name: Shaileshbhai Gothi
Email: s.gothi@ufl.edu

Introduction:

A B+ tree is an N-ary tree with a variable but often large number of children per node. A B+ tree consists of a root, internal nodes and leaves. The root may be either a leaf or a node with two or more children. A B+ tree can be viewed as a B-tree in which each node contains only keys (not key–value pairs), and to which an additional level is added at the bottom with linked leaves.

The primary value of a B+ tree is in storing data for efficient retrieval in a block-oriented storage context — in particular, filesystems. This is primarily because unlike binary search trees, B+ trees have very high fanout (number of pointers to child nodes in a node,[1] typically on the order of 100 or more), which reduces the number of I/O operations required to find an element in the tree.

File Structure:

bplustree.java:

This class contains main method and is therefore the point of entry of the project. It takes the input file name from the command line argument, search for the file, opens it and reads it line by line, performing 4 kind of operations defined by the input file(insert, search by key, search between keys, delete) and writes the output of the searches to a new file, named "output_file.txt".

BPlusTreeImpl.java:

This class implements the B+ tree. The implemented B+ tree takes in order "m" as input and provides four operations insert, search by key, search between keys and delete. In case of duplicate keys, the existing value will be updated.

BPlusTreeNode.java:

This class represents individual B+ tree node. It contains a nested class "Data" that represent the data object of node to store key value pair. It provides lots of utility method to perform operations on B+ tree node.

Function Prototypes:

Class bplustree

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

A java Application to test b+ tree.

Method Summary

Modifier and Type	Method and Description
static void	**main**(java.lang.String[] args) Reads the input file and instantiates a B+ Tree based on provided input, and writes output to file.
private static void	**writeToFile**(java.io.BufferedWriter outputBufferWriter, java.util.ArrayList<java.lang.Double> listValues)

- ***Method Detail***

- **main**

public static void main(java.lang.String[] args)

Reads the input file and instantiates a B+ Tree based on provided input, and writes output to file.

Parameters:
args - The input file name

- **writeToFile**

- private static void writeToFile(java.io.BufferedWriter outputBufferWriter,

-
```
java.util.ArrayList<java.lang.Double> listValues)
                        throws java.io.IOException
```

Parameters:

`outputBufferWriter` - Buffer writer for output file.

`listValues` - List of values found in search.

Throws:

`java.io.IOException`

Class BPlusTreeNode

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

Class to represent B+ tree node.

- ***Nested Class Summary***

Nested Classes

Modifier and Type	Class and Description
(package private) class	**BPlusTreeNode.Data** Nested class to hold B+ tree node data.
(package private) class	**BPlusTreeNode.DataExternalNode** Represents data of external or leaf Node
(package private) class	**BPlusTreeNode.DataInternalNode** Represents data of internal Node

Modifier and Type	Field and Description
private java.util.ArrayList<**BPlusTreeNode**>	**childrens**
private java.util.ArrayList<**BPlusTreeNode.Data**>	**dataList**
private **BPlusTreeNode**	**nextNode**

private BPlusTreeNode	parent
private BPlusTreeNode	prevNode

- ## Constructor Summary

Constructors	
Constructor and Description	
BPlusTreeNode()	
Constructs an empty B+ tree node	

- ## Method Summary

Modifier and Type	Method and Description
void	addChild(int index, BPlusTreeNode node) Adds a child to B+ tree node at give position
void	addData(int index, BPlusTreeNode.Data data) Add data at given index in B+ node
void	addExternalData(int keyIndex, int key, java.lang.Double value) Add new external node data to B+ tree node.
void	addInternalData(int keyIndex, int key) Add new internal node data to B+ tree node.
void	clear() Clear the node
void	clearChildrensList(int fromIndex, int toIndex)

	Clear B+ tree node children list
void	**clearDataList**(int fromIndex, int toIndex) Clear B+ tree node data list
boolean	**containsKeyAtIndex**(int keyIndex, int key) Check if Given key exist in the B+ node at given position
void	**deleteData**(int index) Deletes data at provided index from B+ node
BPlusTreeNode	**getChild**(int index) Get B+ tree node child at provided index
java.util.ArrayList<**BPlusTreeNode**>	**getChildrens**() Get B+ tree node children
java.util.ArrayList<**BPlusTreeNode.Data**>	**getDataList**() Get B+ tree node data
int	**getDataListSize**() Get B+ tree node data list size
int	**getFirstKey**() Get B+ tree node first key in data list
int	**getKeyAt**(int index) Get B+ tree node data
int	**getKeyIndex**(int key) Get the (index+1) child index of given key in dataList
BPlusTreeNode	**getNextNode**() Get Next B+ tree node in linked list for external Node
BPlusTreeNode	**getParent**()
BPlusTreeNode	**getPrevNode**()

	Get Previous B+ tree node in linked list for external Node
boolean	**isOverfullNode**(java.lang.Integer order) Check if Node stores mode data than it is supposed to.
boolean	**isParentOfExternalNode**() Check if B+ tree is parent of External Node.
BPlusTreeNode	**removeChild**(int index) Deletes the child at provided position from B+ node
BPlusTreeNode.Data	**removeFirstData**() Deletes the first data from B+ node
BPlusTreeNode.Data	**removeLastData**() Deletes the last data from B+ node
void	**setChildrens**(java.util.ArrayList<**BPlusTreeNode**> childrens) Set B+ tree node children
void	**setDataList**(java.util.List<**BPlusTreeNode.Data**> dataList) Set B+ tree data list
void	**setNextNode**(**BPlusTreeNode** node) Set Next B+ tree node in linked list for external Node
void	**setParent**(**BPlusTreeNode** node)
void	**setPrevNode**(**BPlusTreeNode** node) Set Previous B+ tree node in linked list for external Node
java.lang.String	**toString**() Override the toString Method to display the data present in node
void	**updateDataValue**(int keyIndex, java.lang.Double value)

	Updates the data value at given index for a B+ tree node
void	**updateKey**(int index, int key) Update the key of B+ node data

Fields

childrens

```
private java.util.ArrayList<BPlusTreeNode> childrens
```

dataList

```
private java.util.ArrayList<BPlusTreeNode.Data> dataList
```

parent

```
private BPlusTreeNode parent
```

nextNode

```
private BPlusTreeNode nextNode
```

prevNode

```
private BPlusTreeNode prevNode
```

Constructor Detail

BPlusTreeNode

```
public BPlusTreeNode()
```

Constructs an empty B+ tree node

Method Detail

getDataList

```
public java.util.ArrayList<BPlusTreeNode.Data> getDataList()
```

Get B+ tree node data

Returns:

the data list of B+ tree node

addChild

- `public void addChild(int index,`

BPlusTreeNode node)

Adds a child to B+ tree node at give position

Parameters:

index - the position where to add child.

node - child to be added.

- **setDataList**

public void setDataList(java.util.List<BPlusTreeNode.Data> dataList)

Set B+ tree data list

Parameters:

dataList - the data list that needs to be set.

- **getParent**

public BPlusTreeNode getParent()

Returns:

Parent node of B+ tree

- **getChildrens**

public java.util.ArrayList<BPlusTreeNode> getChildrens()

Get B+ tree node children

Returns:

the children of B+ tree node

- **setChildrens**

public void setChildrens(java.util.ArrayList<BPlusTreeNode> childrens)

Set B+ tree node children

Parameters:

childrens - ArrayList of children of B+ tree node

- **getNextNode**

public BPlusTreeNode getNextNode()

Get Next B+ tree node in linked list for external Node

Returns:

B+ tree node

- **getPrevNode**

public BPlusTreeNode getPrevNode()

Get Previous B+ tree node in linked list for external Node

Returns:

B+ tree node

- **setParent**

```
public void setParent(BPlusTreeNode node)
```

Parameters:

node - Parent to be set for B+ tree node

- **setNextNode**

```
public void setNextNode(BPlusTreeNode node)
```

Set Next B+ tree node in linked list for external Node

Parameters:

node - The B+ tree external node

- **setPrevNode**

```
public void setPrevNode(BPlusTreeNode node)
```

Set Previous B+ tree node in linked list for external Node

Parameters:

node - The B+ tree external node

- **addExternalData**

```
public void addExternalData(int keyIndex,
                            int key,
                            java.lang.Double value)
```

Add new external node data to B+ tree node.

Parameters:

keyIndex - position where to add external node data

key - the key of data to be added

value - the value of data to be added

- **addInternalData**

```
public void addInternalData(int keyIndex,
                            int key)
```

Add new internal node data to B+ tree node.

Parameters:

keyIndex - position where to add external node data

key - the key of data to be added

- **isOverfullNode**

```
public boolean isOverfullNode(java.lang.Integer order)
```

Check if Node stores mode data than it is supposed to.

Parameters:

```
order - Order of B+ tree
```

Returns:

```
boolean value
```

- **getKeyIndex**

```
public int getKeyIndex(int key)
```

Get the (index+1) child index of given key in dataList

Parameters:

```
key - The key whose index needs to be found.
```

Returns:

```
the index
```

- **updateDataValue**

```
public void updateDataValue(int keyIndex,
                            java.lang.Double value)
```

Updates the data value at given index for a B+ tree node

Parameters:

```
keyIndex - the position of data

value - the value to be updated
```

- **containsKeyAtIndex**

```
public boolean containsKeyAtIndex(int keyIndex,
                                  int key)
```

Check if Given key exist in the B+ node at given position

Parameters:

```
keyIndex - the position of data in B+ node

key - the key to be validated
```

Returns:

```
boolean value
```

- **getDataListSize**

```
public int getDataListSize()
```

Get B+ tree node data list size

Returns:

the size

- **clearDataList**

- ```
  public void clearDataList(int fromIndex,
                            int toIndex)
  ```

  Clear B+ tree node data list

  Parameters:

  fromIndex - starting index from where data list needs to be cleared

  toIndex - ending index till where data list needs to be cleared

- **getFirstKey**

  ```
  public int getFirstKey()
  ```

  Get B+ tree node first key in data list

  Returns:

  the first key

- **isParentOfExternalNode**

  ```
  public boolean isParentOfExternalNode()
  ```

  Check if B+ tree is parent of External Node.

  Returns:

  the boolean value

- **clearChildrensList**

- ```
  public void clearChildrensList(int fromIndex,
                                 int toIndex)
  ```

  Clear B+ tree node children list

  Parameters:

  fromIndex - starting index from where children list needs to be cleared

  toIndex - ending index till where children list needs to be cleared

- **deleteData**

  ```
  public void deleteData(int index)
  ```

  Deletes data at provided index from B+ node

  Parameters:

  index - the position of data

- **getChild**

```
public BPlusTreeNode getChild(int index)
```

Get B+ tree node child at provided index

Parameters:

`index - the index of child`

Returns:

`the B+ tree node`

- **updateKey**

- ```
  public void updateKey(int index,
                        int key)
  ```

Update the key of B+ node data

Parameters:

`index - the index at which data needs to be updated`

`key - the newKey value for update`

- **removeFirstData**

```
public BPlusTreeNode.Data removeFirstData()
```

Deletes the first data from B+ node

Returns:

`the deleted data`

- **removeLastData**

```
public BPlusTreeNode.Data removeLastData()
```

Deletes the last data from B+ node

Returns:

`the deleted data`

- **addData**

- ```
  public void addData(int index,
                      BPlusTreeNode.Data data)
  ```

Add data at given index in B+ node

Parameters:

`index - position at which the data needs to be added`

`data - the data object that will be added`

- **removeChild**

```
public BPlusTreeNode removeChild(int index)
```

Deletes the child at provided position from B+ node

Parameters:
`index - the position of child`

Returns:
`the deleted child`

- **getKeyAt**

```
public int getKeyAt(int index)
```

Get B+ tree node data

Parameters:
`index - the index of key in node`

Returns:
`the data list of B+ tree node`

- **toString**

```
public java.lang.String toString()
```

Override the toString Method to display the data present in node

Overrides:
`toString in class java.lang.Object`

Returns:
`the stringified form of data.`

- **clear**

```
public void clear()
```

Clear the node

Class BPlusTreeImpl

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

B+ tree Impementation. The primary value of a B+ tree is in storing data for efficient retrieval in a block-oriented storage context —in particular, filesystems Note that this implemenation is not snychroized.

Field Summary

Fields

Modifier and Type	Field and Description
`private java.lang.Integer`	`order`
`private BPlusTreeNode`	`root`

- ***Constructor Summary***

Constructors

Constructor and Description
`BPlusTreeImpl(java.lang.Integer order)` Constructs an empty B+Tree or order provided.

- ***Method Summary***

Modifier and Type	Method and Description
`private void`	`addDataToExternalNode(BPlusTreeNode node, int key, java.lang.Double value)` Add a new data to external node
`private void`	`balanceExternalNode(BPlusTreeNode node, int key)` In case of empty External Node we need to balance it using two cases: 1.
`private void`	`balanceInternalNode(BPlusTreeNode node, int key)` In case of empty Internal node we need to balance it using two cases: 1.
`private void`	`borrowFromInternalSibling(BPlusTreeNode node, BPlusTreeNode sibling, int key, boolean isRight)` Case1 of balancing internal node.

private void	**borrowFromLeftSibling**(**BPlusTreeNode** externalNode, **BPlusTreeNode** sibling, int deletedKey) Case1 of balancing external node.
private void	**borrowFromRightSibling**(**BPlusTreeNode** externalNode, **BPlusTreeNode** sibling, int deletedKey) Case1 of balancing external node.
void	**delete**(int key) Deletes an element with given key from the tree.
private **BPlusTreeNode**	**findExternalNode**(**BPlusTreeNode** root, int key) The external node that may contain given key.
private **BPlusTreeNode**	**getLeftSibling**(**BPlusTreeNode** node, int key) Get the left sibling for a node
private **BPlusTreeNode**	**getRightSibling**(**BPlusTreeNode** node, int key) Get the right sibling for a node
void	**insert**(int key, java.lang.Double value) Inserts an element with given key and value in the tree.
private void	**mergeInternalNodes**(**BPlusTreeNode** parentNode, **BPlusTreeNode** newSplitMiddleNode, **BPlusTreeNode** prevSplitNode) Merge two internal Nodes, the generated middle node from split needs to be merged with parent internal node.
private void	**mergeNodes**(**BPlusTreeNode** internalNode, **BPlusTreeNode** newInternalNode, **BPlusTreeNode** prevSplitNode) Recursively Split and Merge internal Nodes, the generated middle node from split needs to be merged with parent internal node.
private void	**mergeWithSibling**(**BPlusTreeNode** externalNode, int deletedKey, boolean isRight) Case2 of balancing external node.

private void	**mergeWithSiblingAndParentKey**(**BPlusTreeNode** node, **BPlusTreeNode** sibling, int deletedKey, boolean isRight) Case2 of balancing internal node.
void	**printBPlusTree**() Prints the complete tree in a human readable format for debugging.
java.util.ArrayList<java.lang.Double>	**search**(int key) Search an element with given key in the tree.
java.util.ArrayList<java.lang.Double>	**search**(int startKey, int endKey) Search all element that lies between and including startKey and endKey
private **BPlusTreeNode**	**splitExternalNode**(**BPlusTreeNode** node) Split external node making the middle key as parent and from middle key to end as child.
private **BPlusTreeNode**	**splitInternalNode**(**BPlusTreeNode** node) Split internal node making the middle key as parent and middle+1 till end as child

- **Methods inherited from class java.lang.Object**

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

-

- ***Field Detail***

- **order**

private java.lang.Integer order

- **root**

private BPlusTreeNode root

- ***Constructor Detail***

- **BPlusTreeImpl**

```
BPlusTreeImpl(java.lang.Integer order)
```

Constructs an empty B+Tree or order provided.

Parameters:

```
order - The order of B+ Tree. Normally an integer greater than 2.
```

- ***Method Detail***

- **insert**

- ```
  public void insert(int key,
                     java.lang.Double value)
  ```

Inserts an element with given key and value in the tree.

Parameters:

```
key - key of the element to be inserted

value - value of the element to be inserted
```

- **delete**

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

Deletes an element with given key from the tree.

Parameters:

```
key - key of the element to be deleted
```

- **balanceExternalNode**

- ```
  private void balanceExternalNode(BPlusTreeNode node,
                                   int key)
  ```

In case of empty External Node we need to balance it using two cases: 1. Borrow from adjacent sibling if they have key's greater than order/2. 2. Merge with sibling and delete the in between key 2. Merge parentKey and sibling

Parameters:

```
node - The internal node that needs to be balanced

key - key which was deleted that caused the imbalance
```

- **balanceInternalNode**

- ```
  private void balanceInternalNode(BPlusTreeNode node,
                                   int key)
  ```

In case of empty Internal node we need to balance it using two cases:
  ```

1. Borrow from adjacent sibling if they have key's greater than order/2 and change parent key from last leaf in case of left sibling or 2nd key in case of right sibling

2. Merge parent and sibling.

Parameters:

node - The internal node that needs to be balanced.

key - The key which caused the imbalance.

- **borrowFromRightSibling**

```
private void borrowFromRightSibling(BPlusTreeNode externalNode,
                                    BPlusTreeNode sibling,
                                    int deletedKey)
```

Case1 of balancing external node. Borrow the first key from right sibling.

Parameters:

externalNode - the node that is getting balanced

sibling - The right sibling which has key greater than order/2

deletedKey - the key that got deleted and caused imbalance.

- **borrowFromLeftSibling**

```
private void borrowFromLeftSibling(BPlusTreeNode externalNode,
                                   BPlusTreeNode sibling,
                                   int deletedKey)
```

Case1 of balancing external node. Borrow the last key from left sibling.

Parameters:

externalNode - the node that is getting balanced

sibling - The left sibling which has key greater than order/2

deletedKey - the key that got deleted and caused imbalance.

- **mergeWithSibling**

```
private void mergeWithSibling(BPlusTreeNode externalNode,
                              int deletedKey,
                              boolean isRight)
```

Case2 of balancing external node. Delete in between key from parent and remove the external node.

Parameters:

externalNode - the node that is getting balanced

deletedKey - the key that got deleted and caused imbalance.

isRight - true in case of merging with right sibling.

- **borrowFromInternalSibling**

- `private void borrowFromInternalSibling(BPlusTreeNode node,`
- `BPlusTreeNode sibling,`
- `int key,`
  `boolean isRight)`

Case1 of balancing internal node. Borrow a key from sibling internal node and change parent key from last leaf in case of left sibling or 2nd key in case of right sibling.

Parameters:

`node - The internal node that is getting balanced.`

`sibling - The sibling which has key greater than order/2`

`key - the key that got deleted and caused imbalance.`

`isRight - true in case of right sibling`

- **mergeWithSiblingAndParentKey**

- `private void mergeWithSiblingAndParentKey(BPlusTreeNode node,`
- `BPlusTreeNode sibling,`
- `int deletedKey,`
  `boolean isRight)`

Case2 of balancing internal node. Make the parent key as in between key of internal node.

Parameters:

`node - The internal node that is getting balanced.`

`sibling - The sibling which is getting merged with parent.`

`deletedKey - the key that got deleted and caused imbalance.`

`isRight - true in case of right sibling`

- **getRightSibling**

- `private BPlusTreeNode getRightSibling(BPlusTreeNode node,`
  `int key)`

Get the right sibling for a node

Parameters:

`node - B+ node whose right sibling is needed.`

`key - key that got deleted from node.`

Returns:

`The right sibling`

- **getLeftSibling**

- `private BPlusTreeNode getLeftSibling(BPlusTreeNode node,`
  `int key)`

Get the left sibling for a node

Parameters:

node - B+ node whose left sibling is needed.

key - key that got deleted from node.

Returns:

The right sibling

- **search**

```
public java.util.ArrayList<java.lang.Double> search(int key)
```

Search an element with given key in the tree.

Parameters:

key - key of the element to be searched.

Returns:

list containing the value whose key is searched.

- **search**

- ```
  public java.util.ArrayList<java.lang.Double> search(int startKey,
                                                      int endKey)
  ```

Search all element that lies between and including startKey and endKey

Parameters:

startKey - starting Key of the element to be searched.

endKey - ending key of the element to be searched.

Returns:

list of values between startKey and endKey

- **findExternalNode**

- ```
  private BPlusTreeNode findExternalNode(BPlusTreeNode root,
                                         int key)
  ```

The external node that may contain given key.

Parameters:

root - Root of B+ tree

key - The key that needs to be found.

Returns:

The external node.

- **addDataToExternalNode**

- ```
  private void addDataToExternalNode(BPlusTreeNode node,
  ```
- ```
                                     int key,
                                     java.lang.Double value)
  ```

Add a new data to external node

Parameters:

`node - The external node.`

`key - The key of data.`

`value - The value of data.`

- **splitExternalNode**

```
private BPlusTreeNode splitExternalNode(BPlusTreeNode node)
```

Split external node making the middle key as parent and from middle key to end as child.

Parameters:

`node - The external node`

Returns:

`The middle node.`

- **splitInternalNode**

```
private BPlusTreeNode splitInternalNode(BPlusTreeNode node)
```

Split internal node making the middle key as parent and middle+1 till end as child

Parameters:

`node - The internal node`

Returns:

`The middle node.`

- **mergeNodes**

```
private void mergeNodes(BPlusTreeNode internalNode,
                        BPlusTreeNode newInternalNode,
                        BPlusTreeNode prevSplitNode)
```

Recursively Split and Merge internal Nodes, the generated middle node from split needs to be merged with parent internal node.

Parameters:

`internalNode - The parent internal node`

`newInternalNode - The middle node`

`prevSplitNode - The split node`

- **mergeInternalNodes**

```
private void mergeInternalNodes(BPlusTreeNode parentNode,
                                BPlusTreeNode newSplitMiddleNode,
                                BPlusTreeNode prevSplitNode)
```

Merge two internal Nodes, the generated middle node from split needs to be merged with parent internal node.

Parameters:

`parentNode - The parent internal node`

`newSplitMiddleNode - The middle node`

`prevSplitNode - The split node`

- **printBPlusTree**

```
public void printBPlusTree()
```

Prints the complete tree in a human readable format for debugging.

References:

https://en.wikipedia.org/wiki/B%2B_tree