Assignment 1
Advanced Data Structures Report on B Plus Trees

Author : Karan Chauhan (16408235)
Email Id : karanchauhan@ufl.edu

1. 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. [1]

The data structure that is often used in the implementation of database indexes. Each internal node of the tree contains an ordered list of keys, and pointers to lower level nodes in the tree. To search for or insert an element into the tree, one loads the root node, finds the adjacent keys that the searched-for value is between, and follows the corresponding pointer to the next node in the tree. Recursing eventually leads to the desired value or the conclusion that the value is not present. [2]

B+ trees use a balancing technique to make sure that all of the leaves (external nodes) are always on the same level of the tree, and that each node is always at least half full of keys. Therefore the height of the tree is always at most $\lceil (\log(n)/\log(k/2)) \rceil$ + c where n is the number of values in the tree and k is the maximum number of keys in each block and c is a constant. This hints that only a small number of pointer traversals are essential to search for a value if a node has a large number of keys. This is crucial in a database because the B+ tree is on disk. [2]

Finally, the leaves of the B+ tree are maintained in a linked list and therefore contain a "next sibling" pointer for fast iteration through a contiguous block of values. This allows for extremely efficient range queries (find the block containing the first value and blast through siblings until the last value is found).

2. Directory Structure

```
.
├── com
│  └── karan
│     └── ads
│        └── bplustree
│           ├── BPlusTree.class
│           ├── BPlusTree.java
│           ├── Key.class
│           ├── Key.java
│           ├── Node.class
│           └── Node.java
├── input.txt
├── Makefile
├── output_file.txt
├── treesearch.class
└── treesearch.java
```

Note: The class files here are indicative of the location where the make file will compile the Java files to.

3. Class Structure

The project uses four classes to implement the B-Plus tree:

3.1 Key

This class defines a key value pair to facilitate storage of duplicate keys. If duplicates are inserted, then the new values are inserted to the same key (maintained as a list) instead of creating a new key value pair. The values are set only for the external nodes, and not for internal nodes, in compliance with the property of B Plus Tree. The class also contains getters and setter methods to fetch and update the corresponding parameter.

Field	Type	Description
key	double	Key of an element of the B Plus tree
value	List<String>	List of values for the key

3.2 Node

This class defines the node structure for a B Plus Tree. The ‘keys’ parameter stores a list of key values pairs in the node. The children of the node are stored in the ‘children’ parameter. These are set only if the node is an internal node, since external nodes have no children. The ‘next’ and ‘prev’ parameters are used to maintain the linked list property of the external nodes. Again, these are set only for the external nodes, since only the external nodes are part of a linked list. All nodes have a parent field, which points to the node's parent. The parent for the root is set to null. The class also contains getters and setter methods to fetch and update the corresponding parameter.

Field	Type	Description
keys	List<Key>	List of keys in a node
children	List<Node>	Children of this node. Set only for internal nodes
prev	Node	Previous element of linked list. Set only for external nodes
next	Node	Next element of linked list. Set only for external nodes
Parent	Node	Parent of node. Null for root

3.3 BPlusTree

This class defines the methods to initialize a b plus tree, insert an element to it, search for a key in the tree, and search for all key value pairs between two keys. Helper methods to print the tree by level using a breadth first traversal, and to do a binary search within a node (to find the correct position where the key would be inserted in the node) have been also defined.

Field	Type	Description
m	int	Stores the degree of the B Plus tree
root	Node	Maintains the root of the B Plus tree

3.4 treesearch

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

4. Function Prototypes

4.1 BPlusTree

4.1.1 `public void initialize(int order)`

This method initializes the B Plus Tree. Sets the degree of the B-Plus Tree as the input order.

Field	Type	Description
order	int	The degree of the B Plus tree

4.1.2 `public void insert(double key, String value)`

This method is used to insert a new key value pair to the B Plus Tree. Three cases are considered here. If the B Plus tree is empty, a new node is created, set as the root, and its key is set as the newly inserted key. Secondly, if the B Plus tree has only one node which is not overfull, the new keys are update to the root node. The last case is a normal insert, where we first traverse to the external nodes, split if overfull, merge with the parent and carry the split upwards till we find a non-overfull node. This is achieved using the helper functions. Note: for traversing to the next level, we use the binary search helper method to find the index of that child of the node, and hence subtree, where this key should go to.

Field	Type	Description
key	double	The key to be inserted
value	String	The value to be inserted

4.1.3 `private void insertWithinExternalNode(double key, String value, Node node)`

This method finds the index where the key would appear if inserted in a sorted list using the binary search helper method. If the key already exists, then the value is added to the list of values for the key. If the key does not exist, then a new pair is inserted at the index found.

Field	Type	Description
key	double	The key to be inserted
value	String	The value to be inserted
node	Node	The external node where the pair will be inserted

4.1.4 `private void splitExternalNode(Node curr, int m)`

This method splits the external node using the b plus tree algorithm to split an overfull external node: the middle element's index is calculated using the formula, m/2, that divides the node into two almost symmetric parts, the left part containing all the elements till m/2 -1, and the right part containing the elements from the index m/2 to the end of the list. The middle element is made the parent of the right part and is later merged with the parent internal node, using the helper method splitInternalNode.

Field	Type	Description
m	int	The degree of the B Plus tree
curr	Node	The overfull external node that has to be split

4.1.5 `private void splitInternalNode(Node curr, Node prev, int m, Node toBeInserted, boolean firstSplit)`

This method takes into account two cases. If the node previously split was itself the root, then the middle element propagated by that split is made the new root, and what's left in the previously overfull node becomes the left child of the new root. The linked list is also updated. The second case takes a more general approach, first we merge the middle element of the previous split with the current node, and merge the children using the helper method mergeInternalNodes. Thereafter, if the merged node is within the b plus node size constraint, we are done. If it becomes overfull again, we use the B tree algorithm to split an overfull node, to split the internal node. In this case, the middle element is at the index $\lceil (m/2) \rceil - 1$ and the left part contains elements including the index 0 up until the index of the middle element (exclusive); the right part contains elements starting from the element after the middle element until the end of the list. The list of children of the left and right nodes are updated. Those children whose keys are greater than the middle element are moved from the overfull node (now the left part) to become the children of the right part. Lastly, the middle element and the split is propagated one level up using recursion.

Field	Type	Description
curr	Node	The current node
prev	Node	The previously split node (child of curr)
m	int	The degree of the B Plus tree
toBeInserted	Node	The middle part of the previous split that has to be merged with the current node.
firstSplit	Boolean	Indicates if the internal node is a parent of an external node

4.1.6 `private void mergeInternalNodes(Node mergeFrom, Node mergeInto)`

This method is used to logically merge the middle part of an over-full node - mergeFrom (with the right part as its child) with its parent - mergeInto. Firstly, using binary search, the index is found where the key of the mergeFrom should be inserted in the sorted key list of the mergeInto. Next, the right part is now inserted into the children list of the mergeInto at its correct position, using the index of the inserted key. Lastly, if the mergeFrom's children are external nodes, then the linked list also needs to be updated. We take into consideration whether the position of insertion of the node in the linked list is the first index of the linked list or the last, and correspondingly update the left and right pointers of the previous and next elements.

Field	Type	Description
mergeFrom	Node	The node to be merged from (middle part of the previously split node)
mergeInto	Node	The node to be merged to (Parent of the previously split node)

4.1.7 `public void printTree()`

This a helper method to print the tree using a level order traversal.

4.1.8 `public void printNode(Node curr)`

This a helper method to print a node for the B Plus tree.

4.1.9 `private void binarySearchWithinInternalNode(double key, List<Key> keyList)`

This method is used to do a modified binary search within an internal node. It returns the first index if the key is less than the first element, and the size of list + 1 if the key is greater than the last element. Thereafter we ensure that we find a location such that the key is smaller than the element at that index and greater than or equal to the element at the previous index. This ensures that we have found a location that should be the correct position where the key would be placed, if it were inserted in the sorted keyList.

Field	Type	Description
key	double	The key whose correct index needs to be found
keyList	List<Key>	The sorted list of keys where the modified binary search needs to be executed

4.1.10 `public List<String> search(double key)`

This method is used to search for all the values for a key in the b plus tree. If the key is not found, it returns null. We first start from the root and iterate to the external node using the binary search method and the key to find the correct children subtree at each level. Finally after finding the correct external node, where the key ‘should’ lie in, a linear search is performed in the external node to find the values for the key.

Field	Type	Description
key	double	The key whose values need to be searched for

4.1.11 `public List<Key> search(double key1, double key2)`

This method is used to search for all the key value pairs such that their keys are between key1 and key2. If no such key satisfies the condition, the method returns an empty list. We first start from the root and iterate to the external node using the binary search method and the key1 to find the correct children subtree at each level. On reaching the external node, the linked list is traversed till we reach the end of the list or encounter a key whose value is greater than key2, and keep updating the result if the key criterion is satisified.

Field	Type	Description
key1	double	The starting index
key2	double	The ending index

4.2 treesearch

4.2.1 `public static void main(String args[])`

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

Field	Type	Description
args	String	Command Line Arguments

4.2.2 `private static BufferedWriter openNewFile() throws IOException`

This method is used to create a new file for writing output to.

4.2.3 `private static void writeSearchByKey(List<String> res, BufferedWriter bw) throws IOException`

This method is used to write the result of search by key to the output file in the prescribed format.

Field	Type	Description
res	List<String>	Result containing values for the key
bw	BufferedWriter	Points to the output file

4.2.4 `private static void writeSearchByKeys(List<Key> res, BufferedWriter bw) throws IOException`

This method is used to write the result of the search of key value pairs between two keys to the output file in the prescribed format.

Field	Type	Description
res	List<Key>	Result containing key value pairs
bw	BufferedWriter	Points to the output file

5. Citations

1. https://en.wikipedia.org/wiki/B%2B_tree
2. https://www.quora.com/What-is-a-B%2B-Tree/answer/Michael-Harris?share=8f97718a