COP5536: Advanced Data Structures

Programming Assignment Report

Submitted By:

Name: Mohit Gupta
UFID: 90991438
mohit.gupta@ufl.edu

Problem statement:

Develop and test a class B+Tree to store pairs of the form (key, value). For this project you are to implement a memory resident B+tree (i.e., the entire tree resides in main memory). Your implementation must be able to store multiple pairs that have the same key (i.e., duplicates). The leaves should be linked into a doubly linked list to support efficient range retrieval. The supported operations are:

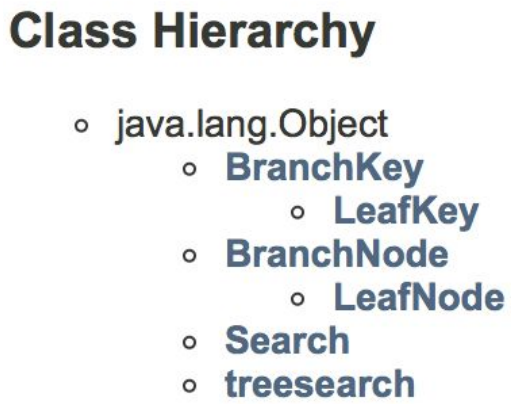
1. Initialize(m): create a new order m B+Tree
2. Insert (key, value)
3. Search (key) : returns all values associated with the key
4. Search (key1,key2): returns (all key value pairs) such that key1 <= key <= key2.

Implemented Solution in Java:

A B+ tree implementation with bottom up splitting approach has been done in JAVA for the given problem statement.

Following are the results from the implementation:
- Leaves are doubly linked
- The B+ tree has been programmed to initialize only when order >=3.
- B+ tree supports Initialize(m)
- B+ tree supports Insert(key, value)
- B+ tree supports search(key) returns all associated values
- B+ tree supports search(key1, key2) returns all key value pairs in the range [key1,key2]

Class Hierarchy for project:



Java Files in the project and their significance

File: treesearch.java

-Class with main method
-Reads source file from command line arguments
-Outputs single key and range search results to file output.txt
-If the degree is <3, does not initialize the tree and quits

Methods:

main(String args[])
initialize()

File: BranchNode.java

-Class with branch nodes handling for the B+ tree
-Checks whether after insert and leaf node splitting, is branch node splitting required
-Splits the overfull branch nodes when needed
-Manages the left and right pointers of the branch node keys

Constructors:

public **BranchNode**(BranchNode branchNode)

BranchNode()

Methods:

void **splitAdjustNode**(BranchNode newBranchNode, ArrayList<BranchNode> traceback)

BranchNode **adjustParentNode**(BranchNode branchNode, BranchNode singleKeyNode)

boolean **isSplitRequired**(ArrayList<TreeKey> treeNodeKeys)

void **fixMiddleKeyPointers**(BranchNode newRightNode, BranchNode updatedParent, ArrayList<TreeKey> leftList, ArrayList<TreeKey> rightList, TreeKey newParentKey)

File: Search.java

-Class with search functionality for the b+ tree
-Searches for single key for input command search(key)
-Searches for keys within a range for input command search(key1, key2)
-Formats the range search result to comma separated pairs. [(key,value),(key1,value1)...] pairs

Methods:

String **rangeSearch**(LeafNode leafNode, float firstKey, float secondKey)

String **singleKeySearch**(LeafNode leafNode, float key)

String **formatRangeSearch**(LeafKey leafKey)

File: LeafNode.java

-Class that handles leaf nodes
-Handles new key,value pair insert
-If tree is empty, inserts to root
-Finds the correct leaf node to insert in the tree to maintain the B+ tree
-Inserts the new key in ascending order in the suitable leaf node
-Checks if key already exists in the tree, if yes, appends the value to current key for further usage when searching the key with multiple values.

Constructors:

LeafNode(LeafNode leftLeafNode, LeafNode rightLeafNode)

Methods:

boolean **keyExists**(float key)

void **insertNode**(float key, String val)

LeafNode **findLeafNode**(float key, TreeNode root)

File: LeafKey.java

-Class that handles the keys inside a leaf node
-Checks if split is required in the leafnode after insertion or not

Constructors:

LeafKey(String dataVal,float keyVal)

Methods:

boolean **isSplitRequired**(ArrayList<LeafKey> leafKeys)

void **AddLeafKey**(ArrayList<LeafKey> leafKeys, LeafKey newLeafKey, LeafNode leafTreeNode)

File: BranchKey.java

-Class that manages the keys in the Branch nodes
-Useful for managing left and right pointers for Branch nodes after insert

Constructors:

BranchKey(double keyValue, BranchNode leftNode, BranchNode rightNode)

BranchKey(double key)

Testing

Code has been thoroughly tested using various set of inputs and different sequences.

Single Key test cases checked for:
-Empty key
-Key exists in tree with single value
-Key exists in tree with multiple values
-Key does not exist in the tree

Range Key test cases checked for:
-Both keys exist in tree
-One key exists in tree
-None of the keys exist in tree

Orders for which B+ tree is tested:
-3,4,5,6,7
-Root value and node values verified against manual tree structure for smaller input set

Technologies Used:

Java Version used: 1.8
IDE used: IntellJ Community version