COP 5536 Fall 2017

Programming Project – Implementation of B-Plus Tree

Ravi Teja Poloju

3917 – 3690

rpoloju@ufl.edu

Created a B+ Tree to store key value pairs where key is of type “Float” and value is of type “String”.

Programming language used: Java

Operations Implemented:

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.

Input file format required: First line of the file must have the order of B+ tree to be created followed by Insert/Search Operations holding the format - Insert(<key>,<value>) Search(<key>) Search(<value1>,<value2>)

The zip file contains makefile and .java files with no nested directories.
To run: *java TreeSearch input.txt*
The output file named “output_file.txt” will be generated in the same folder.

Implementation details:

Initially a B+ tree of given order is created with a single leaf node. Once it exceeds the capacity, it is split halfway forming tree nodes and leaf nodes.

All the nodes (except leaf nodes) are of type TreeNode and the leaf nodes are of type LeafNode.

The program flow is as follows.

Initially an empty LeafNode is created with the given order. The classes TreeNode and LeafNode extend Node class. TreeNode and LeafNode are vectors containing DataNode elements. The DataNode is a float value.

LeafNode.insert(DataNode) – takes DataNode as an argument, finds the position for the element to be inserted and inserts. It returns the root of the tree.

TreeNode.insert(DataNode) – this returns the parent of the leaf that contains the element required while search. Searching is important for correct insertion of the elements.

LeafNode.split(DataNode) – When the leaf node is full, this function splits the leafnode. First the item is inserted into the vector and then split halfway (at n/2 for even ‘n’ or n+1/2 for odd ‘n’). Then the next nodes are set for these leaf nodes since all the leaf nodes form a linked list. Then the last element of the first split is taken and propagated up towards root.

Node.propagate(DataNode, Node) – It has 3 cases – parent could be null, parent could be full, or parent is available with some space. When parent is null, a new parent is created with the data propagated up and pointers are updated. When parent is not full, the node propagated up is inserted in its correct location and pointers are updated. When the parent is full, then the parent is split like the leaf node split. Here the parent of all the leaves is updated, since parent changes due to TreeNode split.

TreeNode.getPointerTo(DataNode) – returns the parent of a leaf node/tree node that contains this DataNode.

TreeNode.getPointerAt(int) – returns the Node at the specified index from leaf node/tree node.

TreeNode.search(DataNode) – returns the leaf node where the item is located.

LeafNode.setNextNode(LeafNode) – sets the next Node since all the leaf nodes form a linked list.

LeafNode.getNextNode() – returns the next Node. Returns null for last node.

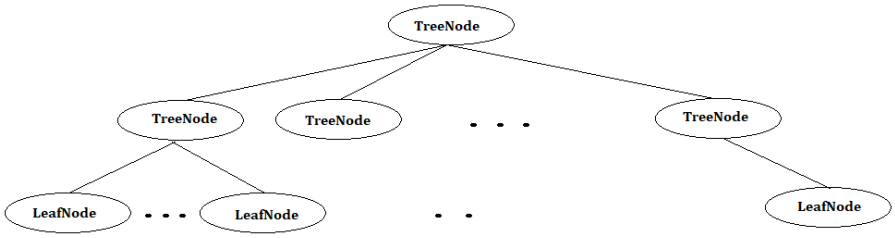
LeafNode.search(DataNode) – iterates over the leaf node and returns true if found.

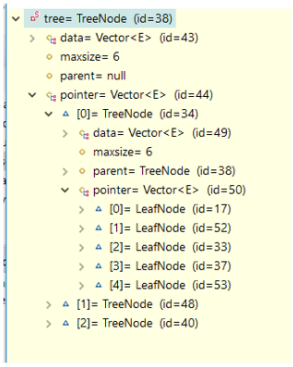
DataNode.getData() – returns the key(float value) from DataNode object.

DataNode.inOrder(DataNode) – returns true if the key contained in the DataNode coming from the argument is less than the current key.

CustomMap – created a map like structure using hash to store the key value pairs so that retrieving the value with the given key takes O(1) time since Search must be taken place in O(log n) time.

Structure:





Output for the given sample “input.txt” file:

```
Value41

(-
0.31,Value84)(0.89,Value42)(1.04,Value50)(15.52,Value73)(22.75,Value48)(26.72,Value49)
(27.37,Value9)

Value113, Value149, Value184, Value212

(-28.83,Value99)(-28.74,Value100)(-20.28,Value125)(-13.78,Value86)(-12.82,Value199)(-
8.95,Value222)(-4.66,Value47)(-3.84,Value207)(-
0.31,Value84)(0.89,Value42)(1.04,Value50)(15.52,Value73)(17.99,Value170)(22.75,Value48
)(25.29,Value139)(26.72,Value49)(27.37,Value9)(34.58,Value186)(36.57,Value226)(37.58,V
alue168)(37.78,Value71)(39.46,Value164)(42.02,Value23)(44.15,Value133)(46.7,Value103)(
48.68,Value135)(54.56,Value60)(54.74,Value213)(56.49,Value132)

Value7219

Null

Value9957, Value9983

Value9952

Null
```