Advanced Data Structure Project Report
Memory Resident B+ Tree

Sagar Arora UFID-14476353 arorasagar1811@ufl.edu Fall2017

***Abstract*—In this report, we will discuss the implementation of B+ Trees. The scope of this report is limited to use such the keys and values from input text file. the system. 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.**

Keywords—insertion, splitNode, LeafNode, InternalNode, splitInternalNode, B+ tree, search,

I. INTRODUCTION B+ TREES

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.

PROGRAM STRUCTURE AND COMPILING PROCEDURE

There are total five files: KeyVaue.java, Node.java, InternalNode.java, LeafNode.java, Tuple.java, BplusTree.java,

treesearch.java.

The makefile included in the zip uses javac compiler and creates the necessary classes. To run the program, type

java treesearch filename

treesearch contains the method to parse the filename and contains the method to format the output according to the requirements.

II. NODE.CLASS

Node.class is parent class of InternalNode.class and LeafNode.class. In abstract sense, it defines the structure of both InternalNode.class and LeafNode.class. A Node class has following attributes:

1) boolean isleafNode – defines weather the node is Leaf Node or InternalNode.

2) List keys – list of keys for each node.

III. INTERNALNODE.CLASS

InternalNode.class is subclass of Node. An InternalNode class has following attributes:

1) boolean isleafNode – defines weather the node is LeafNode or InternalNode., because both the nodes have their own methods and atributes.

2) List keys – list of keys for each node inherited from Node.class

3) List children – list of children. Children can be a LeafNode or InternalNode.

4) public void insertSorted(Tuple<Double,Node> e, int index)

Parameter: Tuple, index Return Type: void

The insertSorted method is implemented to find the location of Tuple in the list of the Tuples, to put it at that location in the list.

IV. LEAFNODE.CLASS

LeafNode.class is subclass of Node. An InternalNode class has following attributes:

1) LeafNode prevNode: since the Leaf Node also forms the linked list, prevNode points to the previous Leaf Node.

2) LeafNode nextNode: since the Next Node also forms the linked list, prevNode points to the next Leaf Node.

3) KeyValue keyVal: Refrence to the KeyValue class. It contains a key and multiple String values. This is provided for Duplication of values on a single key. for each key there is Arrakeys aylist of Values.

4) List<KeyValue> keyvalues: After each keyvalue is created, it is added in the List of the particular leafnode.

4) public void insertSorted(KeyValue k)

Parameter: Tuple, index Return Type: void

This method sorts the KeyValue pair.

V. TUPLE.CLASS

Tuple.class is a utility class created to assist in creating the Map Entry that is to create a tuple of Key and Value. This class is further used in BplusTree.class to map key and value pairs as needed.

1) public final A a: This variable stores the key.

2) public final B b: This variable stores the Value.

3) public Tuple(A a, B b) :

Parameter: A and B

This is the constructor to intialize the key value pair in the given tuple.

VI. KeyValue.class

KeyValue.class contains a key and multiple String values. This is provided for Duplication of values on a single key. for each key there is Arraylist of Values.

1) double key: stores the key of the given pair.

2) List<String> values: stores the values associated with each key. It can also be duplicate. If duplicate is there it is added in this list.

3) public void setValue(String value)

4) public void insertSorted(KeyValue k)

Parameter: KeyValue, index Return Type: void

VII. BplusTree.class

BplusTree.class is the main class with all the logic for search, insertion, range search written. It has following attributes and methods:

1) public Node root: points to the root node.

2) public static int D: D is the order of the B+ Tree. It is initialized by the user.

3) public List<String> search(double key): Search recursively looks for the key in the B+ Tree. It calls subsidiary method treeSearch to find the node for which the key is being searched. If no leaf node is found, search sends the null, other wise linear search is done to find out the value associated with the key.

4) public Node treeSearch(Node node,double key):

5) public List<Tuple<Double,List<String>>> search(double key1, double key2):

Parameter: double key1, double key2

Return Type: List<Tuple<Double,List<String>>>

first the search is applied on the key1. If the leaf node for the key1 is found, leaf node is traversed to find the values greater than the key1 and all the values lesser than the key2. If end of leaf node is reached than by using the next pointer (since we are maintaining the doubly linked list) we traverse the next node by it and again compare the key with key1 and key2.

6) public void insert(double key, String value):

Parameter: double key Return Type: String value

The leaf pages are maintained in sequential order and a doubly linked list connects each leaf page with its sibling page(s). This doubly linked list speeds data movement as the pages grow and contract. Do a search to determine what keyValue pair in which the new record should go in if the record is already available. If not then a new keyValue pair is inserted into the appropriate leaf. if the leaf is not full, add the keyValue, otherwise, split the LeafNode. Allocate new leaf and move half the leaf's elements to the new leaf node and insert the new leaf's smallest key and address into the parent. if the parent is full, split it also now add the *middle* key to the parent node repeat until a parent is found that need not split.if the root splits, create a new root which has one key and two pointers.

7) public Tuple<Double, Node> splitLeafNode(LeafNode leaf)

Parameter: LeafNode Return Type: Tuple<Double, Node>

As we add a KeyValue containing a key to the leaf node, leaf node becomes full, so the method splitLeafNode is called by the insert method. A new Leaf Node is created and all the keyValue pairs are transferred from the previous leaf node to this new leaf node. The middle key from the keyvalue pair is sent to the Index node. We will Allocate new leaf and move half the leaf's elements to the new leaf node and insert the new leaf's smallest key and address into the parent. if the parent is full, split it also now add the *middle* key to the parent node repeat until a parent is found that need not split, if the root splits, create a new root which has one key and two pointers.

8)public Tuple<Double, Node> splitInternalNode(InternalNode index)

Parameter: InternalNode Return Type: Tuple<Double, Node>

If the node contains fewer than the maximum allowed number of elements, then there is room for the new element. Insert the new element in the node, keeping the node's elements ordered.

Otherwise the node is full, evenly split it into two nodes so:

A single median is chosen from among the leaf's elements and the new element.

Values less than the median are put in the new left node and values greater than the median are put in the new right node, with the median acting as a separation value.

The separation value is inserted in the node's parent, which may cause it to be split, and so on. If the node has no parent (i.e., the node was the root), create a new root above this node (increasing the height of the tree).

VIII. CONCLUSION

Output of the program matches for all the input files provided on canvas. For smaller number of inputs, program runs for around 100ms. I also tested the program for input file containing million tags and the runtime was around 3.7