ADVANCE DATA

STRUCTURES (COP5536)

SPRING 2019

PROGRAMMING PROJECT REPORT

SUBMITTED BY

NAME: VIKAS PATHAK
UFID: 7198-4931
Email ID: pathak.vikas@ufl.edu

B+ Tree implementation

Background

A B+ tree is balanced tree structure in which the index nodes direct the search and the leaf nodes contain the data entry.

Assignment was to implement the following functions as well as a description of the implementation:

1. Initialize (m): create a new m-way B+ tree
2. Insert (key, value)
3. Delete (key)
4. Search (key): returns the value associated with the key
5. Search (key1, key2): returns values such that in the range key1 <= key <= key2

Key should be Integer Type and Value should be Float/Double type.

COMPILING INSTRUCTIONS

- The project has been written in JAVA and can be compiled using standard JDK 1.8.0_60, with javac compiler.
- Unzip the submitted folder. The folder contains the project files (.java file), the Make file.
- The input text files must be placed in the project directory before running the program.
- To execute the program, we can remotely access the server using ssh username@thunder.cise.ufl.edu
- For running B+ tree on an input file, type:
- To compile:
- javac bplustree.java
- To execute:

- java bplustree input

Steps to execute the project assignment using make file

1. Copy the source java file, input file and the make file in a separate directory.

2. Reach to that directory through terminal.

3. Copy the data file (if not done)

4. Run make.

Now, the project is ready to be executed.

STRUCTURE

CLASS bplustree :

- This class contains the main function from where the program execution starts (entry point).
- The input file is parsed as an argument and output files are created.
- The input file is parsed and according to the given operations appropriate function calls are called.
- An object of a B+ tree is created with the parsed order i.e.*tree.*
- The whole parsing and file handling is done with the help of "Buffered Reader Class and its Objects" present in IO package.
- Exceptions are handled by throws keyword by help of Exception class.

CLASS Node:

- Creating a class Node with 2 variables and 2 functions.
    1. *keys*– Integer ArrayList of the keys of the tree.
    2. *isLeafNode* – Boolean variable to check whether given object of node is leaf or not.
    3. *isOverflowed() and isUnderflowed()*- Boolean Function to check whether given treeobject size node is greater than or less than respectively from order of tree inputted .

CLASS IndexNode:

- Each IndexNode is having Node class type ArrayList of child to store child keys and new children. Following the B+ plus tree rule of atleast two children.
- insertSorted() function is to insert the entry into this node at the specified index so that it still remains sorted.

CLASS LeafNode:

- This class is an extension to Node where the inserted data is stored.
- For storing, Arraylists data structure is used into values variable.
- Each leafNode has nextLeaf pointer to store nextLeadNode.
- insertSorted() function is same as in IndexNode to insert key/value into this node so that it still remains sorted.

CLASS BPTree

Creating a class BPTree with 2 variables

root – Root of our tree of Node type

m – Order of our tree.

Initialize(int) function: This function is used for initializing the BPlus Tree with degree specified in the input file.

Search

Logic: A key is given ,function has to return the associated double value. This was implemented recursively to find the leaf node containing the number. Then a traversal through the leaf node was used to find the key. If the key is not found, it returns null.

Function Prototype:

public String search(int key)

private LeafNode SearchLfNodeandKey(Node theNode, int key)

-This function finds the LeafNode where the is to be inserted to.

Search Range

Logic: In this function, search for the key1 in the same way as it has been

done in the previous function (search(key)).First find recursively the leaf node containing the matching key.

Here our range can go from one leafnode to other. Therefore, after the end of each leafnode, using next pointers, check all keys of the next leaf and iteratively do the search that meet the last occurrence of key which lies between key1 and key2 (both inclusive).

Function Prototype:

public String search(int key1, int key2)

private LeafNode SearchLfNodeandKey(Node theNode, int key)

Insert

Logic: When the root is null, created a new leaf node with a (key, value) entry, else, searched the leaf node to insert that entry and handle possible overflow. Therefore,created a helper function called "insertHelper" which does the following things:

- Recursively find the leaf node to be inserted.
- If the leaf node is found, insert the given entry.
- When the current root in this function is going to be overflowed, handle it using either splitLeafNode or splitIndexNode.
- Return the new Entry object after dealing overflow.

Function Prototype:

public void insert(int key, double value)

Function Calls:

private Entry<Integer, Node> insertHelper(Node node, int key, double value)

-*This function is Helper to insert() method.*

public Entry<Integer, Node> splitIndexNode(IndexNode position)

-This function split an indexNode and return the new right node and the splitting key as an Entry<splitingKey, RightNode> (key/node pair)

public Entry<Integer, Node> splitLeafNode(LeafNode leaf)

-This function split a leaf node and return the new right node and the splitting key as an Entry<splitingKey, RightNode> (key/node pair)

private void ResortSibPointers(LeafNode leftLeaf, LeafNode rightLeaf)

-This function rearranges pointers for the leftLeaf's nextLeaf pointer.

Delete

The deletion is like the inverse process of insertion, so the code structures is quite same. Therefore created a helper function called "deleteHelper" function.Three conditions arises as follows:

- Recursively find the leaf node to delete the given key.
- If the leaf node is found, find the key and delete the entry.
- When the current root in this function is going to be underflowed, deal it using either manageLeafNodeUnderflow or manageIndexNodeUnderflow.
- Return the splitting index to the previous caller, so that we can keep the tree well-structured.

But the node underflow is typical to manage than the overflow since either leaf node or index node could be merged or redistributed with its siblings. So, a parent is added and the corresponding child index to the Node class. Because it makes easier to find the siblings of a node.

Function Prototype:

public void delete(int key)

Function Calls:

private int deleteHelper(IndexNode parent, Node node, int key)

-This function is Helper to delete() method.

public int manageLeafNodeUnderflow(LeafNode left, LeafNode right,IndexNode parent)

-This function handles LeafNode Underflow (merge or redistribution)

-splitkey position in parent if merged so that parent can delete the splitkey later on,or else -1.

public int manageIndexNodeUnderflow(IndexNode leftIndex,IndexNode rightIndex, IndexNode parent)

-This function handles IndexNode Underflow (merge or redistribution).

-splitkey position in parent if merged so that parent can delete the splitkey later on,or else -1.

private void UpdateIndxNodeKeyandKey(Node theNode, int searchKey, int newKey)

- *This function is used to update ancestors which could contain a deleted key.*