ADVANCED DATA STRUCTURES

COP 5536
Spring 2019

Programming Project

Submitted by:
Name: Prakhar Mittal
UFID: 3909-9969
Email: prakharmittal@ufl.edu

Introduction

A **B+ tree** is n-ary tree which 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.

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. A Recursing function eventually leads to the desired value or the conclusion that the value is not present. B+ trees use a balancing technique to make sure that all of the leaves are always on the same level as 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.

We were asked to develop and test a small degree B+ tree used for internal-memory dictionaries.
The data is given in the form (key, value), we were required to implement an m-way B+ tree to store the data pairs. In a B+ tree only leaf nodes contain the actual values, and the leaves should be linked into a doubly linked list. A degree is defined as the maximum number of child nodes which can exist at a point. If during the insertion, the number of nodes exceed the degree, then that node ought to spilt and rearrange all the nodes between parent and child to maintain the degree of B+ tree.

The implementation should support the following operations:
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

Function Prototypes

1. **public static void main(String args[])throws IOException**
   This is the place where the actual project executes(point of entry). It takes the file an expected input file, appends ".txt" and hence performs the corresponding 'insert', search(key) and search(key,key2). The output is stored as "output_file.txt". An IOException to avoid any ambiguity or out of bounds exception in the project.

2. **private static BufferedWriter newFile() throws IOException**
   Creates new file to write the output.

3. **private static void searchKey(List<Double> request, BufferedWriter buffer) throws IOException**
   This method is used to write the result of search by key to the output file in the prescribed format.

4. **private static void key_search(List<Bp_key> request, BufferedWriter buffer) throws IOException**
   Enables the range search result format required for the project.

5. **Class Node**
   This class is used for the definition of the node structure for the B+ tree. It involves the internal_child and the node_parent parameters. The next and prev(for previous) are also declared to manage the linked list.

6. **public void initialize(int order)**
   This function is used to initialize the B+ Tree. It takes the input order m to set its degree

7. **public void node_insert(int key, double value)**
   This function is used here to insert the (key,value) pair. It is broken into three possible scenarios: If the B Plus tree is empty, a new node is created, and set as the root. If the B Plus tree has only one node, the new keys are update to the root node. The last case is a standard insert, where we traverse to the last node and split if full.

8. **private void insert_ExternalNode(Node node, double value, int key)**
   If a list is sorted already using a Binary search, this function finds out the index value of where the key would reside. A new pair is inserted if a pair already isn't present.

9. **private void ext_nodeSplit(int m, Node node_current)**

   This function will split the node which is full. The split is performed via a standard m/2, which first divides the nodes into 2 parts. The middle element hence is made the parent node.

10. **private void internal_NodeSplit(Node prev, int m, Node node_Insert, boolean firstSplit, Node node_current)**

    When the node that was recently split was a root, then the middle element is made the parent and the remaining part is made the left child of the tree.In the other case we take a more standard way were we merge the left child and the parent node using the function mentioned below.

11. **private void internal_NodeMerge(Node interal_NodeMerge, Node interal_NodeMergeResult)**

    This function is used to actually merge the otherwise full node ie the children and the parent of the internal node. This function incorporates binary search to determine the first index value for 'internal_NodeMerge' will insert the values. After the left, the right part is now merged into the node.

12. **public int search_NodeInternal(int key, List<Bp_key> keyList)**

    This function is used to change the binary search function on the internal node. Hence, after changing the index value to index+1 we can now find the element smaller than the index value.Hence a sorted list would then be inserted in the node.

13. **public int Delete_Node(int key, List<Bp_key> keyList)**

    This function takes the requested value from the input file and uses the corresponding search to find the key. The key is then set to Null and the internal_NodeMerge is invoked to merge the remaining node structure of the B+.tree

14. **public List<Bp_key> node_search(int key1, int key2)**

    This function is used to return the (key,value) pairs such that they are between key1 and key2. If no such pair exists, it returns an empty linked list.

15. **public List<Double> node_search(int key)**

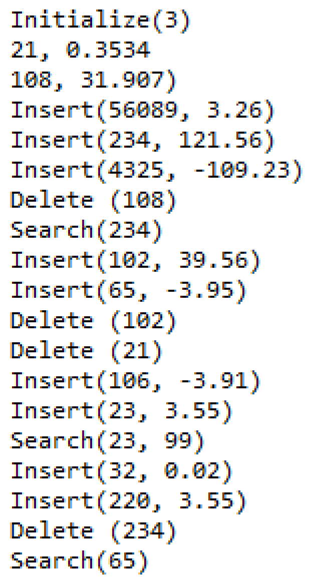
    This function is used to search for all the values corresponding to the keys in the B+ tree. When the iterator reaches the desired leaf, the result is returned.

Programming Environment

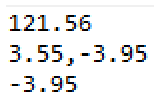
Software:
Programming Language: Java v1.8

Hardware:
Operating Systems: Windows 10
RAM: 8GB
Hard Disk: 1TB

Sample Snapshots of Input and Output:



```
Initialize(3)
21, 0.3534
108, 31.907)
Insert(56089, 3.26)
Insert(234, 121.56)
Insert(4325, -109.23)
Delete (108)
Search(234)
Insert(102, 39.56)
Insert(65, -3.95)
Delete (102)
Delete (21)
Insert(106, -3.91)
Insert(23, 3.55)
Search(23, 99)
Insert(32, 0.02)
Insert(220, 3.55)
Delete (234)
Search(65)
```



```
121.56
3.55,-3.95
-3.95
```