Neharika Khera

UFID:

89500993

neharikakhera@ufl.edu

Project Description

The project revolves around the implementation of B+ trees. B+ tree is a data structure used for storing Key, Value pairs. The attribute of B+ tree which differentiates it from other data structures is the same level of its leaf node. The challenge of implementing B+ tree is to sustain its required degree at each node. 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 assignment was focused on four main functionalities:

1. Degree Initialization: The first & foremost step was to initialize the degree 'm' of the B+ tree

2. Insert Key Value Pair: There could be any command possible after initialization. If its Insertion, then the Insert( ) function will be called up and the Key Value pair will be inserted at the desired place.

3.Search Key: This search ( ) functionality deals with the traversing of whole B+ tree. It requires to traverse the B+ tree node by node with the help of given key & displays the associated values as the output.

4. Range Search: In this case, user searches for a particular range of key values as (Key1,Key2) . The expected output is the all key value pairs associated with this range such that Key1 <=Key <=Key2

Programming Environment

Software:

Programming Language: Java

Hardware:

Operating Systems: Windows 10

RAM: 8GB

Hard Disk: 256 GB

Project structure

The implementation of B + tree has been done in Java language.

1. Source Code: It is divided into three java files: First file i.e. BPlusTree.java contains all methods to be implemented i.e. initialize(), search(), insertPair(), leaf merge() ,nonleafsplit(), parentmerge(), getSearch() , printSearch(), rangeSearch() . These function help in performing task of Insertion, Search and SearchRange in B+ tree. The second file i.e. treesearch.java contains main function . Data is read from input file and and functions from BPlusTree.java are called accordingly. The third file is FileHandler.java where all file handling methods are operated like FileInputStream and BufferedReader.

2. Input File : The input file named 'input.txt' is an another important element of this project. This file's first line contains the initialize(order) command to initialize tree by specifying order of the tree . Later on, it gives any of the three commands
    - Insert(Key,Value): It invokes the Insert function & performs the insertion at the desired location
    - Delete(Key): It invokes the delete function & delete the key at leaf.
    - Search(Key): This input element invokes the getSearch ( ) function which in turn invokes the search ( ) function. It searches for the associated value with respect to the key.
    - Search(Key1, Key2): This search operation again invokes the rangeSearch ( ) function which in turn invokes the search( ) function & looks for the Key1 position. Since, the nodes are interconnected with each other, rangeSearch ( ) function directly displays the all the node values until it encounters the Key 2 position.

3. Output File: This is a file named 'output_file.txt' where all the output values will be generated and will be written on it once the whole program is executed

4. Make File: This file will consists of the addresses of the required files and hence will be used to execute the program

Structure

```
public class BPlusTree{

        int m; //the order of tree
        int n; //minimum number of required children/children
```

```
public Queue<Node> q = new LinkedList<Node>();

 public class Node{

            public boolean isleaf;
            public boolean isroot;

            //number of keys
              public int nokeys;

            //keys
            public double[] keys = new double[m

            //value node to handle duplicates
            public Vnode[] values = new Vnode[m];

            //children
            public Node[] children = new Node[m+1];//can be at max m, and adding a padding element

            //children for doubly linked list
             public Node next;
             public Node previous;

            //parent
            public Node parent;

            Node(){}

             Node(boolean isleaf, boolean isroot){
                        this.isleaf = isleaf;
                        this.isroot = isroot;
                         }

               }

        Node root;
```

FUNCTIONS

Functions Used:

1. Initialize(order)

2. void InsertPair (double key, double value)

3. void delete(double key)

4. void leafmerge (Node leaf, int key, double value)

5. void nonleafsplit (Node nonleaf, Node emptynode)

6. void parentmerge (Node oldlefthalf, Node newrighthalf, double key)

7. void Node search(double key, Node node)

8. void getSearch(double Key, Node node)

9. String printSearch (Vector<String> result)

10. String rangeSearch(double key1, double key2)

11. void main(String args[])

Functions Detailed Working

1. main (String args[])

Description: The main function runs to initialize all the parameters. It creates BPlusTree by making its object.

It further reads the input file and calls the respective function according to the formal arguments. Possible function could be:
insertPair (int key, double value) rangeSearch (int key1, int key2)
getSearch (int key1)

Input Parameter: The input argument is the input filename or the absolute path of the file.

2. insertPair (double key, double value)

Description: This function is invoked when there are two formal arguments in the insert method. The first argument is of type integer and another is of type double.

To check the precise location of the insertion, we have to call the search ( ) function. search( ) gets us a leaf node where value associated with key would probably be inserted . Then check whether the returned node is already present or it. If the key is already present then for that already present key, values are stored. Else if key is not already present, we check for number of keys in a node and merge into leaf accordingly.

Pseudo Code:

```
leaf = Search (key, root)
if keyalreadypresent
    insert into valuenode
else
    if leaf.no_keys< capacity
       mergeIntoLeaf
    else
        mergeIntoLeaf
```

newLeaf created
inserting 2nd half data into new leaf
mergeIntoParent

3. mergewithleaf (Node leaf, int key, double value)

Description: Finds index in leaf node where key could be inserted. Then values associated with keys and keys are inserted.

Pseudo Code
i=whereToinsert
s hifting and making space for key and its value

3. NonLeafSplit(Node nonleaf, Node emptyNode)
Description: nonleaf node has greater than m-1 keys, no it needs split. An empty node is filled by taking some keys from nonleaf node along with split key.And then split is also removed from now filled empty node. A nonleaf node, filled empty node and split key are returned.

4. Parentmerge(Node oldnodewithlefthalf, Node newnodewithrighthalf, int key)
Description: We check if oldnodewithlefthalf is root or not. If it is, we allocate new root and put new leaf's first key in it. And if oldnodewithlefthalf is not a root, we find where key could be inserted in parent and then then check overflow condition.

Pseudo Code
If(oldnodewithlefthalf .isroot)
new root created
else
i=wheretoinsert
if overflow
splitnonleaf
parentmerge

5. Node search(double key, Node node)
Description: It performs search operation on the tree based on key and returns a leaf node where either key exist or key-value pair could be added.

Pseudo Code

```
if key exist in node
    if leaf
    return node
    else
    return search(key, next pointer for node)
else if key < node . key[i]
        if leaf && pointer == NULL
          return node
        else
          return search(key, node pointer)
```

6. getSearch(double key, Node node)
   This method uses printsearch method to return the perfectly formatted string on search operation on single key.

   Pseudo Code

```
Calls search(key, node)
if keyexpected == key
  ispresent = true
  value is extracted for the key
else
    result=NULL
return result
```

7. printSearch(String result)
   This method prints the result obtained from getSearch( ) for search operation in single key.

8. rangeSearch(int key1, int key2)
   searches for node where key1 could be present and finds the values associated with the key less than or equal to key2 in continuing doubly linked list in leafnodes. Then these values are printed.

   Pseudo Code

```
Leaf = search(key1,node)
Result = values from key1 to key2 are found
if result isempty
   Result= NULL
else print result obtained
```

Compiling and Running the project

1. Login to server ‘thunder.cise.ufl.edu’ using putty or an equivalent Unix emulator with the cise username and password

2. Run Make File to compile java application

3. Run java command with input file as an argument

4. Output file i.e. output_file.txt is created