ISHITA TRIVEDI

UFID: 6893-6496

itrivedi@ufl.edu

Project Description:

The purpose of this project is implementation of B+ trees. B+ tree is a data structure consisting of a root, internal nodes and leaves. It is mainly used for storing Key and Value pairs. The attribute of B+ tree which differentiates it from other data structures is that it stores data pointers at the leaf nodes. The leaf nodes form the first level of indexing, and the internal nodes form the other levels of a multilevel index. 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 implements five main operations:

1. Initialize (m): This creates a B+ tree of degree 'm'.

2. Insert (Key, Value): This inserts the Key and the value associated with the key in the B+ Tree.

3. Search (Key): This search ( ) function traverses the B+ tree. It searches if the 'Key' is present in the tree and if present, returns the value associated with it. If the key is not present, it returns NULL.

4. Search (Key1, Key2): This operation returns all the values of the keys in a given range. The output is all key value pairs within range Key1 <=Key <=Key2.
5. Delete (Key): The Delete () function deletes the node value associated with the Key.

Programming Environment

Software:

Programming Language: C++

Hardware:

Oracle VM box- Linux OS

Project structure

The project has been implemented in C++

1. Source Code:

The project is divided into 3 main .cpp files: BPTree.cpp, BPTNode.cpp and main.cpp. There are also two .hpp files: BPTree.hpp and BPTNode.hpp.

- BPTree.cpp & BPTree.hpp: BPTree.hpp contains declaration of a class BPTree and its variables and functions. BPTree.cpp contains the actual written functions.
  It includes definition and declaration of the following B+ Tree operations:

  - void Initialize (int m)

  - void Insert (int key, float value)

  - void Delete (int key)

  - bool Search (int key, float &value)

  - bool *Search (int key1, int key2, float *&values, int&num)

  -BPTnode *search_Recursive (BPTnode *node, int key)

- BPTNode.cpp & BPTNode.hpp: BPTNode.hpp contains declaration of classes BPTnode, InNode (internal node) and Lnode (leaf node). BPTNode.cpp contains code for the class functions.
  The functions are:

  - Class BPTnode: insert, search, delete, merge, split, getChildren, getValues

  - Class InNode: insert, Delete, merge, split, getChildren

  - Class LNode: insert, Delete, merge, split, getValues

- main.cpp: main.cpp contains the main function. It reads the input file, processes it, carries out the various B+ tree operations and creates an output file with respect to the input.

2. Input File:

   The input file named 'SampleInput.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 5 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 consists of the commands to run all the files and hence will be used to execute the program. The program can be executed by simply typing ‘make’ in the terminal.

Class Structure:

There are two base classes: BPtree and BPTnode. The classes InNode and LNode are derived from class BPTnode.

Following are the classes with their variables and function prototypes:

- class BPTree

```
private:
    int order;
    BPTnode *root;
    BPTnode *search_Recursive(BPTnode *node, int key);
public:
    BPTree ();
    ~BPTree ();
    void Initialize(int m);
    void insert(int key, float value);
    bool search(int key, float &value);
    bool *search(int key1, int key2, float *&values, int &num);
    void Delete(int key);
```

- class BPTnode

```
protected:
    int order;
    int *keys;
    int keyNum;
    bool isLeaf;
    BPTnode *parent;
    BPTnode *prev;
    BPTnode *next;
public:
    BPTnode(int n);
    ~BPTnode();
    bool IsLeaf();
    int *GetKeys();
    int GetKeyNum();
    void decKeyNum();
    int GetKeyIndex(int key);
    BPTnode *GetParent();
    BPTnode *GetNext();
```

```
BPTnode *GetPrev();
void SetParent(BPTnode *node);
void SetNext(BPTnode *node);
void SetPrev(BPTnode *node);
virtual void insert(int key, float value) {}
virtual void insert(int key, BPTnode* rtChild) {}
virtual void insert(int key, BPTnode* lftChild, BPTnode* rtChild) {}
virtual void Delete(int key) {}
virtual void merge(BPTnode* rtNode) {}
virtual void search(int key) {}
virtual void search(int key1, int key2) {}
virtual BPTnode *split(int &key) { return NULL; }
virtual BPTnode **getChildren() { return NULL; }
virtual float *getValues() { return NULL; }
```

- class InNode : public BPTnode

```
private:
    BPTnode **children;
public:
    InNode(int n);
    ~InNode();
    void insert(int key, BPTnode* rtChild);
    void insert(int key, BPTnode* lftChild, BPTnode* rtChild);
    void Delete(int key);
    void merge(BPTnode* rtNode);
    BPTnode *split(int &key);
    BPTnode **getChildren();
```

- class LNode : public BPTnode

```
private:
    float *values;
public:
    LNode(int n);
    ~LNode();
    void insert(int key, float value);
    void Delete(int key);
    void merge(BPTnode* rtNode);
```

```
BPTnode *split(int &key);
float *getValues();
```

Functions and their working:

1. main (String args[])
   The main function opens the input file, makes an instance of the class BPTree so that its functions can be called.

   It further reads the input file and calls the respective functions according to the formal arguments.

2. Initialize(int m):
   The initialize function initialises the order of the B+ tree as m from the input file.

3. insert (int key, float value)
   This function is invoked when there are two formal arguments in the insert function. The first argument is of type integer and another is of type double. The insert function first searches for the leaf node where the value associated with the key can be inserted. It checks whether the returned node already exists and is not full, the value is inserted. Otherwise, split the node, move the middle key to the parent, and insertthe new node to the parent. This keeps moving up the levels till it finds a parent that does not need to split.

4. *search_Recursive(BPTnode *node, int key)
   Insert function uses this function to recursively search for node and key. If pointer of the node is a leaf node, return it. If the pointer is not a leaf node get the key. If key is found, move pointer to the right, if not found, move pointer to the left.

5. Delete(int key)
   Case 1: Delete from a leaf node (This is done by Delete function of class LNode)
   - If there is more than the minimum number of keys in the node. Simply delete the key
   - There is an exact minimum number of keys in the node. Delete the key and borrow a key from the immediate sibling. Add the median key of the sibling node to the parent.

Case 2: Delete from internal node as well (This is done by Delete function of class InNode)

- If there is more than the minimum number of keys in the node, simply delete the key from the leaf node and delete the key from the internal node as well.
- If there is an exact minimum number of keys in the node, then delete the key and borrow a key from its immediate sibling (through the parent).
- Here, empty space is generated above the immediate parent node.After deleting the key, merge the empty space with its sibling.

6. search(int key, float &value)
   This function searches if a value corresponding to the key is present in the B+ Tree and if present, returns the value. If not present, returns NULL.

7. search (int key1, int key2, float *&values, int &num)
   First this function gets the nodes associated with the key 1 and key 2. It then gets the index of these nodes and starts retrieving values between this range.
8. *split(int &key)
   This function is used while inserting a value in a node. This is used to split a node into two when it reaches maximum order.

   A node has greater than m-1 keys, no it needs to split. An empty node is filled by taking some keys from node along with split key. And then split is also removed from now filled empty node. The node, filled empty node and split key are returned.

9. merge(BPTnode* rtNode)
   This function I used to merge nodes in the delete operation when they reach minimum occupancy.

Other small functions:

10. IsLeaf(): returns if node is a leaf
11. GetKeyNum(): returns keyNum
12. GetKeys(): returns keys in a node
13. GetKeyIndex(int key): returns the key index
14. decKeyNum (): Decreases keynum by 1
15. GetParent (): returns the parent of a node
16. GetNext(): returns next value of a node

17. GetPrev(): returns previous value
18. SetParent (): Sets the parent of a node
19. SetNext(): sets the next variable of a node
20. SetPrev():sets the next variable of a node
21. getValues(): It is for the leaf nodes to return the values
22. getChildren(): It is for the internal nodes to return their children

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. Open the directory which contains the project folder.
3. Run Make File to compile all the files in the project: make
4. Run command for inputfile: ./bplustree SampleInput.txt
5. Output file i.e. output_file.txt is created