COP 5536 - ADVANCED DATA STRUCTURES
SPRING 2019

PROGRAMMING PROJECT ON B+ TREES

Submitted by:
Name- Kshitij Raj
UFID - 13584965
Email- kshitijraj@ufl.edu

Project Summary

This project required us to implement a B+ Tree Data Structure. This is a data structure that has a basic skeleton of storing a {key, value} pair ONLY AT THE LEAF NODE, unlike other trees which allows us to store data on non-leaf (internal nodes).

In a B+ Tree, each node has a Degree or Order which states the number of children it can have. The point of concern is that the order of the node cannot exceed the set value else the concerned node has to be split.
The split causes the node to split in two and the new nodes are joined with the parent of the current node and if the order criteria exceeds there, split again and join with its parent. This goes on recursively

Implementation

1] Initialize the order of the B+ Tree using the first line of the input file.

2] Insert the new key value pair in the tree. This calls a function that inserts the new key value pair In the tree based on the key after traversing the tree on checking the key values of the nodes.

3]Delete the key value pair in the tree. This calls the delete function that finds the node with the key that is to   be deleted, deletes it and updates the necessary pointers.

4] Search operation- This operation takes the key value to search, calls the search function that traverses the tree based on the key values in the nodes and returns the value of the corresponding values associated with the key.

5] Search in Range operation- This operation calls the range search function that searches within a key pair and returns all values associated with the two key pairs.

Project Structure

The whole project is made up of 4 files which are:

- Source Code
- Makefile
- Input File
- Output File

1] Source Code contains all the required code for the B+ Tree implementation. It contains all functions like insert(), delete(), search(), search_in_range(), split(), leaf_split(), non-leaf_split() etc.

2] Input file contains the {key,value} pairs of the data to be acted upon like:

- Insert{key,value}
- Delete{key}
- Search{key}
- Search{key1, key2}

3] Output file contains all the output generated when the source code is executed
4] Makefile contains the necessary commands to compile the code.

Program Structure

```
public:
  string greeting;
  int nNodes;
  Node *parentNode;
  Node *next;
  Node *prev;
  vector<tuple<double, string>> record;
  vector<double> keys;
  vector<Node *> children;
  bool isleaf;

  Node(bool leaf = true)
  {
     nNodes = 0;
     parentNode = NULL;
     next = NULL;
     prev = NULL;
     greeting = "alive";
     isleaf = leaf;
  }
};
```

```
Node *rootNode = new Node(); //Create root Node
bool keySort(const Node *a, const Node *b)

void insertNode(Node *curNode, double k, string val)
    bool keySort(const Node *a, const Node *b) //Sort keys
    if (curNode->isleaf) //If leaf Node -> Direct insert
        void splitNode(Node *curNode)
    Else
        Traverse to the appropriate node, -> Insert
        void splitNode(Node *curNode)

void DeleteNode(Node *curNode, double k)
    if (curNode->isleaf) // Delete directly
    & Update all necessary pointers
    Else
        Traverse to the appropriate node, -> Delete

void search(Node *curNode, double k, ofstream &o_file)
    if (!curNode->isleaf)
        Traverse to the appropriate node, -> Write values to output file
    Else (Leaf Node)
        Write to output file directly

void search_in Range(Node *curNode, double k, double l, ofstream &o_file)
    if (!curNode->isleaf)
        Traverse to the appropriate node range -> Write in between values to output file
    Else
        Write range of values to output file directly

int main(int argc, char *argv[])
    ifstream file(argv[1]); //take input file
    ofstream o_file("output_file.txt"); //create output file to write to
    while (getline(file, line))
        Get order of B+ Tree
        if (line.substr(0, 6) == "Insert") //call insert function
        if (line.substr(0, 6) == "Delete") //call delete function
        if (line.substr(0, 1) == "S") //call concerned search function

            Case1
                Search one key and return its value
            Case2
                Search between range of keys and return all values between them
    //Close all open files
    file.close();
    o_file.close();
Exit program
```

Functions explained in detail

1] **void insertNode(double k, string val)**

This function is executed when we want to insert a {key, value} pair in the tree.
We traverse to the required node according to the keys and then insert there.

Pseudo Code:

```
        Insert{key,value}
        If (node!=leaf)
            Return node
        Else
        (node==leaf)
            if(Order<m)
                insert()
            Else
                split()
                insert()
```

2] **void DeleteNode(double k)**

This function is executed when we want to delete a {key} in the tree.
We traverse to the required node according to the keys and then delete there.

Pseudo Code:

```
Delete{key}
If (node==Root)
    Delete Root
Else if (node!=leaf)
    Traverse to leaf Node
    Find the Node with key value
    Delete()
    Update necessary pointers
```

3] **void search(key)**
This function searches for the associated value with respect to the given key value.
This function will traverse the tree based on the key value given and when it finds it, it will return the corresponding values associated with the key or else fall off the tree.

Pseudo Code

```
void search( key )
If (key of current node=key)
    Return node.value
Else if(key of current node<key)
    Go Right
    If (key of current node=key)
    Return node.value
Else
    Go left
    If (key of current node=key)
    Return node.value
```

4] **Search_Range(key1, key2)**

This function searches for the all associated value with respect to the 2 key values.
This function will traverse the tree based on the key values given and when it finds it, it will return the corresponding values associated that lies between the keys.

5] **main ()**
This function calls all the functions depending upon the lines read from the input files.
Further, it reads from the input files and then creates an output file that the program can write to.
It takes an input which is the filename of the input file and the functions possible include:

- Initialize (order)
- insert(key,value)
- delete(key)
- Search(key)
- Search_Range(key1,key2)

6] **Split()**

This function is called when the capacity of a node exceeds its order. That means, it has more children than it can sustain. In that case, we split the nodes and then merge the new-born nodes to the parent of the current node. If the new parent node exceeds its capacity, follow the spilt() again recursively until the order is preserved.

END OF REPORT*