ADVANCED DATA STRUCTURES

COP 5536

Programming Project

Sakshi Dubey

UFID: 48131141

sakshidubey@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,Key 2) . The expected output is the all key value pairs associated with this range such that Key1 <=Key <=Key2

Programming Environment

Software:

Programming Language: C++

Compiler: g++ compiler

Hardware:

Operating Systems: Windows 10

RAM: 8GB

Hard Disk: 256 GB

Project structure

The implementation of B + tree has been done in C++ language. Following are the files which have been used during the whole implementation

1. Source Code: This file contains the entire source code for the implementation. It contains various functions like main ( ), insertnode( ), leaf_splitting( ), nonleaf_splitting , search( ), range search( ). The detailed functionality of each search has been described later.

2. Input File : The input file named 'input.txt' is an another important element of this project. This file's first line contains the degree 'm' of the B+ tree. Later on, it gives any of the three commands

    2.1 Insert(Key,Value): It invokes the Insert function & performs the insertion at the desired location

    2.2 Search(Key): This input element invokes the search ( ) function which in turn invokes the traverse ( ) function. It searches for the associated value with respect to the key.

    2.3 Search(Key1, Key2): This search operation again invokes the traverse ( ) function & looks for the Key1 position. Since, the nodes are interconnected with each other, it directly displays the all the node Values until it encounters the Key 2 position.

3. Output File: This is a file named 'output.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

```
class Proj {
  public:
    int nNodes; //number of nodes
    Proj  *parentNode;
    Proj  *next;
    Proj  *prev;
    vector<tuple<double,string>> record;
    vector<double> keys;
    vector<Node*> children_node;
    bool isleaf;
```

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

FUNCTIONS

Functions Used:

1. int main ( )

2. void InsertNode (Proj * current_node, int val)

3. void leaf_splitting (Proj *current_node)

4. void nonleaf_splitting (Proj *current_node)

5. void search (float Key)

6. void Search (float Key1, float Key2)

Functions Detailed Working

1. main (int m)

Description: The main function runs to initialize all the parameters & it reads the degree 'm' from the 'input.txt' file.

It further reads the input file and calls the respective function according to the formal arguments. Possible function could be:
Insert (float key, int value)
Search (float key)
Search (float key1, float key2)

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

2. insertnode (Proj *current_node, int val)

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

To check the precise location of the insertion, we have to call the traverse ( ) function. Traverse function will tell us if it's a leaf node and if it is, it will check the degree condition requirement.

Pseudo Code:

```
traverse (key, value)
if node is not a leaf
        return node;
else
if node  is equal to leaf
        if degree <m
        insert values
        else
        call split ( ) function
```

Input Parameter: The input parameter contains the key position and the integer value to be inserted.

3. Search (float Key)

Description: This function searches for the associated value with respect to the key position.

This traverse function looks for the exact location of the Key and gives its value as a result.

Input Parameter: The input parameter will contain only the float type Key value which will be picked from input file.

4. Search (float Key1, float Key2)

Description: This function is differentiated by function overloading concept. The type and number of arguments differentiates between both the search.

We definitely need to traverse the whole tree to look for the location of Key1. Once we find this location, we simply jump to the next right nodes with the help of associated pointers/vectors till we arrive to the key 2 position

Pseudo Code:

```
Search (float Key1, float Key2)
traverse ( root);
if key_position is not equal to key1
        root = key node
        traverse(root);
else if key_position is equal to key 1
        print values of key1
        if node->next is not equal to the key2
                node=node->next
                print node_value
```

Input Parameter: This function takes the key value pair as its arguments from the input.txt file.

5. void leaf_splitting (Proj *current_node)

Description: This function is the core of the whole implementation. This function decides when the split is required if the insertion node's degree is equal to the input degree.

Algorithm:

```
if  node's degree is equal to the degree
then split the node from the mid value
```

Input Parameter: The function takes the splitting node's location as the input argument.

6. void nonleaf_splitting (Proj *current_node , int val)

Description: This function is invoked when there is a non leaf splitting.
Input Parameter: Here we pass the present node and the integer value where the splitting is done

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 C++ application

3. Run g++ command with the filename as argument

4. An output file will be generated as follows in output_file.txt: