

ADVANCED DATA STRUCTURES
COP 5536

PROGRAMMING PROJECT
FALL 2017

SUBMITTED BY

NAME: SURBHI JAIN

UFID: 5094-9509

EMAIL: surbhi.jain@ufl.edu

Introduction

The aim was to implement B+ tree to store pairs of the form (key, value) such that we could store multiple pairs that have the same key. The project has been written in C++ and has been compiled using the g++ compiler. A makefile has been also been provided with the source code which runs as (./treesearch input_file.txt) and make clean

Node Structure

For the implementation of the B+ tree we have the structures as:

```
typedef struct key_value
{
    string* value[MAX_DUPLICATES];
    int count;
}
```

```
typedef struct btree
{
    void ** pointers;
    int * keys;
    struct btree * parent;
    bool is_leaf;
    int no_of_keys;
    struct btree * next;
} btree;
```

Struct btree represents a node in B+ tree while struct key_values represents the value a given key refers. The count there stores the number of values we have stored for a particular key. In node struct we have array of keys and array of corresponding pointers.

We have a different relationship between the keys and pointers in leaf and internal nodes. For a leaf, we will have the index of each key equal to the index of its corresponding pointer with maximum key-pointer pairs being order -1. The last pointer

will be NULL if its the rightmost leaf or point to the leaf to the right. However, for an internal node, the lower nodes with keys < smallest key in array is referred to by the first pointer. Then, with j starting at 0, pointer at j+1 points to the subtree with keys >= the key in this node at index j. The last leaf pointer would point to the next leaf;

The is_leaf indicates whether the node is a leaf node or an index node (is_leaf is false for index nodes). The no_of_keys stores the number of keys in the node. num_keys+1 is the number of keys in an internal node. Every node has atmost order-1 keys and at least ceil(order/2). And number of pointers to a subtree would be one more than the number of keys at that time.

Function Prototypes and Description

Function: btree * Insert(btree * root, double key, string value)

Arguments: Root, and the key and value pair to be inserted in the B+ tree.

Return Value: Pointer to node structure where the key was inserted.

Description: Finds if the key already exists and adds to the values list. Otherwise creates a new tree (if root is null) and inserts the key, value pair. In case, key doesn't exist then we search for the node where the key has to be inserted, if the number of keys is less than order-1, we insert in that node, otherwise we split. Initially we start at the root, insert if number of keys is still less than order-1. Else we split the root, which gives us 2 new leaf nodes and a new root. Other

Function: btree * NodeInsertionSplit(btree * root, btree * parent, int left_index, double key, btree * right);

Arguments: The root, node that will be split, the key insertion that caused to split, node that will be the right child and a left index where it will be inserted and then split.

Return Value: The function calls the ParentInsertion function and hence, we get the new root as the return value of this function.

Description: Split the node correctly, inserted the middle value into the parent node and we get the left and right child after the split.

Function: node * ParentInsertion(node * root, node * left, float key, node * right)

Arguments: The root, left node, right node and the key

Return Value: Root after the insertion and splitting.

Description: If parent is null, it creates a new root and adds the appropriate key value to it, else finds an index for insertion and calls the NodeInsertion

if number of keys in parent is less than order -1, otherwise calls NodeInsertionSplit.

Function: void Search(btree * root, double key)

Arguments: Root, key to search.

Return Value: Void.

Description: Obtains the value for the key and Prints the key value pair in the output file.

Function: void Search_range(btree * root, double range1, double range2);

Arguments: root, start key and end key for range search.

Return Value: Void.

Description: Scans through the leaf containing the start key and returns the number of keys found in the range and also stores the keys and values temporarily in a new variable. Prints the keys values pairs then to the output file.


Function: vector<string> ParsingInputFile (string str)

Arguments: String read from input file.

Return Value: vector string containing extracted paramaters from the string.

Description: We find the substring between "(" and ")" initially and then using delimiter as ",", we store it in a vector string, hence getting separate values for each.

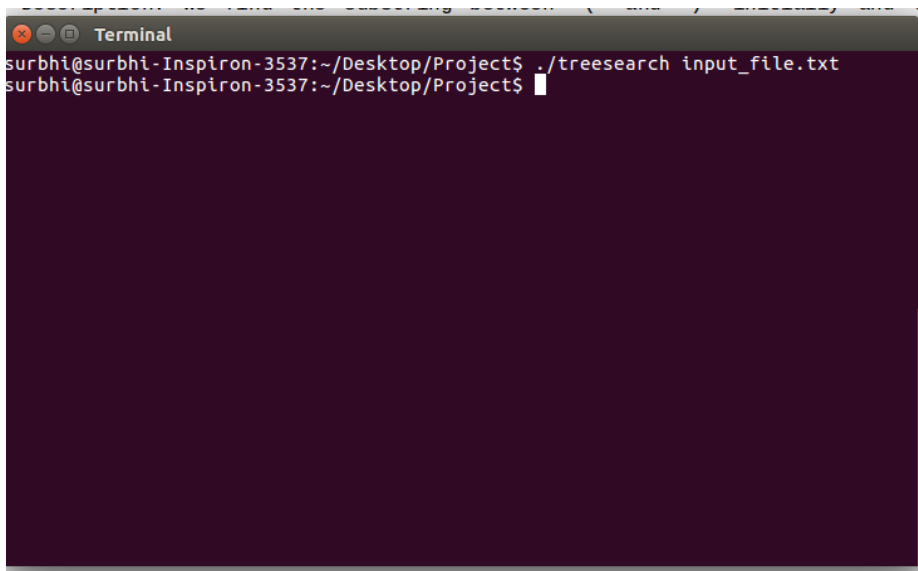
Result Summary



```
output_file.txt (~/Desktop/Project) - gedit
1 val4,val5
2 (2.25,val2) (2.525,val2) (2.9,val2) (4.0001,val4) (4.0001,val5) (4.001,val4) (4.001,val5) (5.08,val5) (5.08,val5) (8.94,val4) (40.5,val4)
  (259.78,val5) (600.9,val6) (788.6,val7)

Plain Text  Tab Width: 8  Ln 1, Col 1  INS
```

Sample output from a sample input file test run.



```
Terminal
surbhi@surbhi-Inspiron-3537:~/Desktop/Project$ ./treeseach input_file.txt
surbhi@surbhi-Inspiron-3537:~/Desktop/Project$
```