PROJECT REPORT ON

B+ TREES

SUBMITTED BY: Suchita S Kanthwal
UFID: 9855-9781
Email: suchitakanthwal@ufl.edu

FOR:
COURSE: Advanced Data Structures (COP5536)

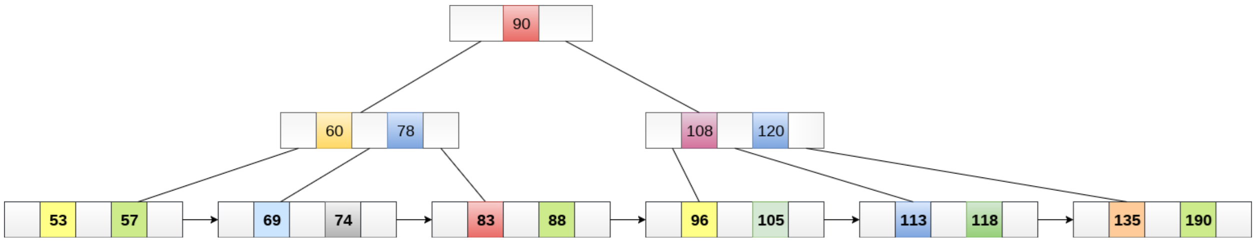
April 14, 2019

Department of Computer and Information Science
Engineering University of Florida, Gainesville, FL
32608

B+ Trees : is an additional part of **B Tree** (a multi way tree) which allows efficient insertion, deletion and search tasks top be efficiently, containing index pages and data pages, where data pages always appear as a leaf nodes in a tree. The root node always comes under index pages. It is basically, N-array tree having large number of children per node in a given tree.

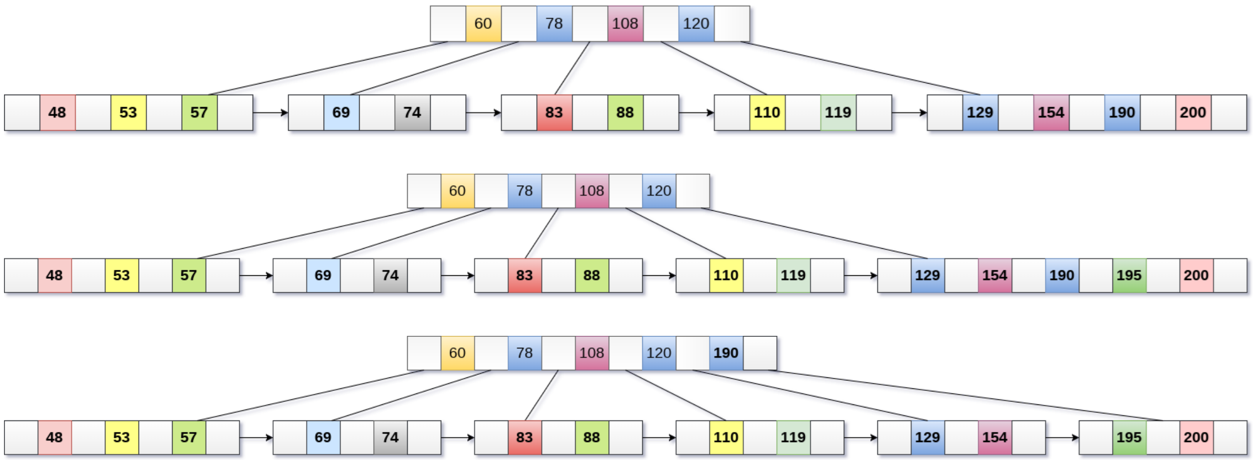
B+ Trees are used to store large amount of data, limiting the fact that size of main memory is always limited, the internal nodes of the B+ trees are stored in the main memory where as there is a special space called secondary memory for leaf nodes.

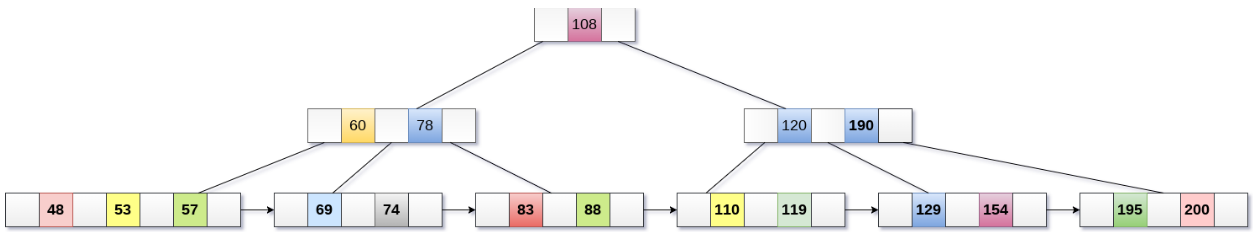
B+ Tree of order three:



Insertion in B+ Tree:
Insert a node as a leaf node.
If leaf do not have any space, then break the existing node and copy the middle node to the next node index.
If in case, index node do not have any left space, then again break the node and copy the middle element to the next index page.

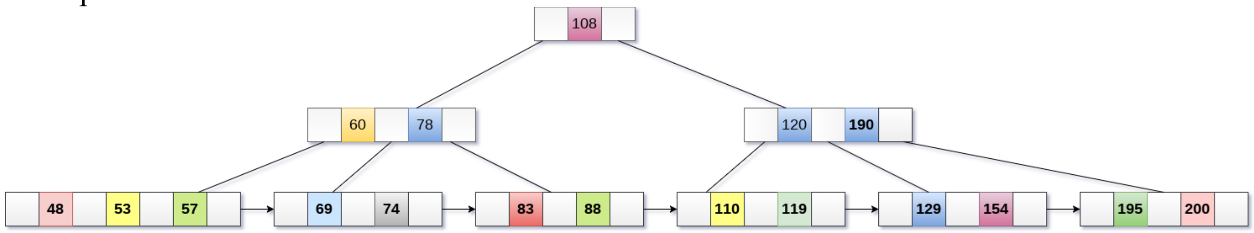
Example:

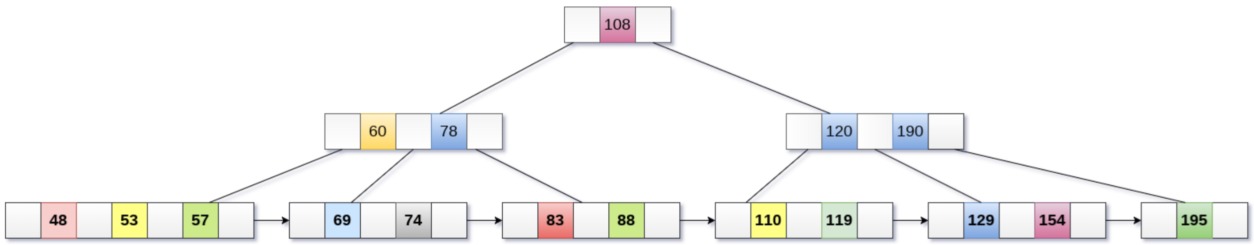


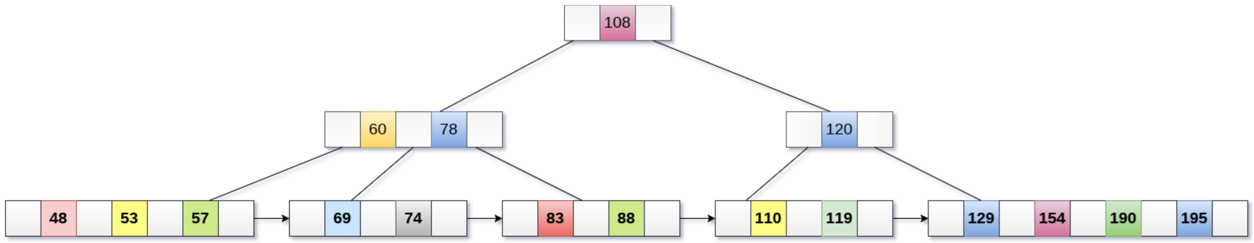


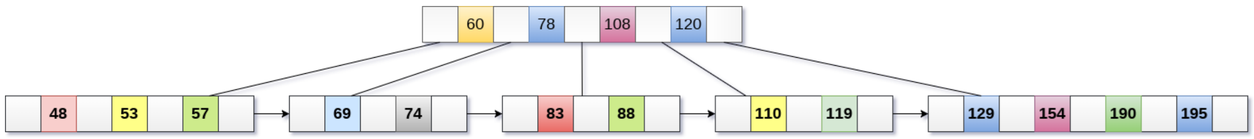
Deletion in B+ Tree:
Delete key from the leaves.
If leaf node contains less numbers than minimum number, join the node with its sibling and delete the key.
If index node contains less than minimum number of elements, join the node with the sibling and move down the key.

Example:









Data Structure and Function capabilities:

```
BPlusTree ();
        ~BPlusTree ();

        void Display ();

        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);


        bool insert();
        void search();
        bool remove();
        void getroot(){
        }
        void maketree();
        int putgloabltime(long gt);
        bool RL();
        bool RR();
        void PrintJobs(int val, int flag, ofstream &result);
        int TravRootnxt(int val, int flag, ofstream &result, int a);
        int TravRootprev(int val, int flag, ofstream &result, int a);
        void PrintJobRange(int start, int end,ofstream &result);
        int PrintJobRangecall(int start, int end, ofstream &result, int a);
        LeafNode (int n);
        ~LeafNode ();

        void Insert (int key, float value);
        void Delete (int key);
        void Merge (BPTNode* rightNode);
        BPTNode *Split (int &key);
        float *GetValues ();
void Insert (int key, BPTNode* rightChild);
        void Insert (int key, BPTNode* leftChild, BPTNode* rightChild);
        void Delete (int key);
        void Merge (BPTNode* rightNode);
        BPTNode *Split (int &key);
        BPTNode **GetChildren ();
```

Final Functioning:

Our program is designed in 3 parts. The main part looks at the input file then calls the b+tree which in turn calls the nodes in the tree to perform all the steps. The structure is as defined in any b+ tree and node have a key to value connection. These mimics the storage in a database. In the end the retrieval of all values is really fast. This has allowed us to set up a good data structure for a database.