B+ Tree

A **B+ Tree** is primarily utilized for implementing dynamic indexing on multiple levels. Compared to B- Tree, the B+ Tree stores the data pointers only at the leaf nodes of the Tree, which makes search more process more accurate and faster.

Rules for B+ Tree:

Here are essential rules for B+ Tree.

- Leaves are used to store data records.
- It stored in the internal nodes of the Tree.
- If a target key value is less than the internal node, then the point just to its left side is followed.
- If a target key value is greater than or equal to the internal node, then the point just to its right side is followed.
- The root has a minimum of two children.

Why use B+ Tree?

Here, are reasons for using B+ Tree:

- Key are primarily utilized to aid the search by directing to the proper Leaf.
- B+ Tree uses a "fill factor" to manage the increase and decrease in a tree.
- In B+ trees, numerous keys can easily be placed on the page of memory because they do not have the data associated with the interior nodes. Therefore, it will quickly access tree data that is on the leaf node.
- A comprehensive full scan of all the elements is a tree that needs just one linear pass because all the leaf nodes of a B+ tree are linked with each other.

Insert Operation:

The following algorithm is applicable for the insert operation:

- 50 percent of the elements in the nodes are moved to a new leaf for storage.
- The parent of the new Leaf is linked accurately with the minimum key value and a new location in the Tree.
- Split the parent node into more locations in case it gets fully utilized.
- Now, for better results, the center key is associated with the top-level node of that Leaf.
- Until the top-level node is not found, keep on iterating the process explained in the above steps.

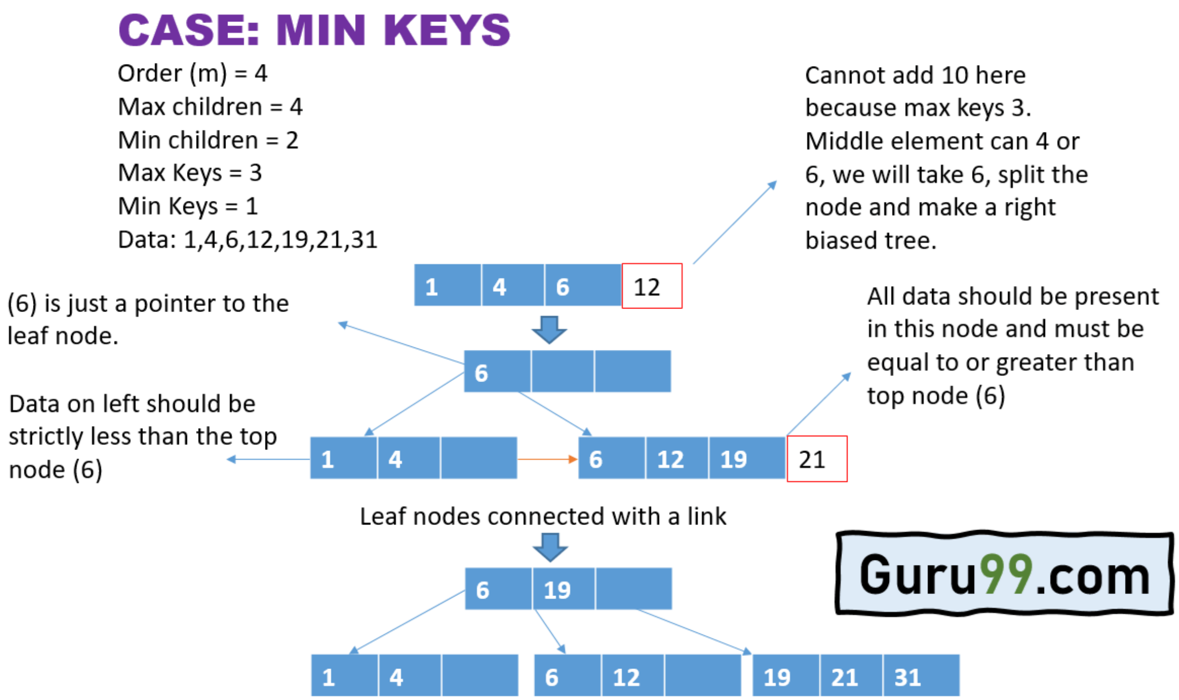
Insert Operation Algorithm:

```
1.      Even inserting at-least 1 entry into the leaf container does not make it ful
l then add the record
     2. Else, divide the node into more locations to fit more records.
      a. Assign a new leaf and transfer 50 percent of the node elements to a new plac
ement in the tree
      b. The minimum key of the binary tree leaf and its new key address are associat
ed with the top-level node.
      c. Divide the top-level node if it gets full of keys and addresses.
          i. Similarly,  insert a key in the center of the top-level node in the hier
archy of the Tree.
     d. Continue to execute the above steps until a top-level node is found that does
not need to be divided anymore.

3) Build a new top-level root node of 1 Key and 2 indicators.
```

Output:

The algorithm will determine the element and successfully insert it in the required leaf node.



The above B+ Tree sample example is explained in the steps below:

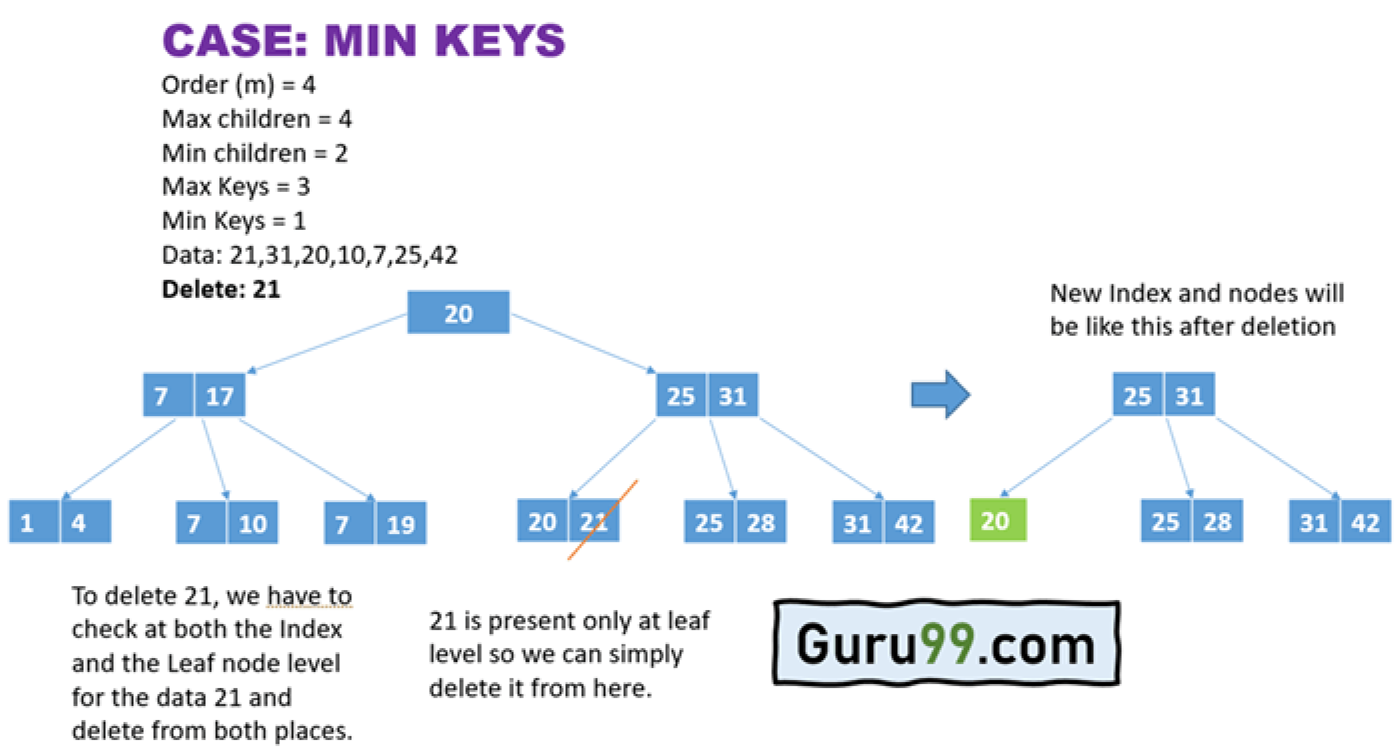
- Firstly, we have 3 nodes, and the first 3 elements, which are 1, 4, and 6, are added on appropriate locations in the nodes.
- The next value in the series of data is 12 that needs to be made part of the Tree.
- To achieve this, divide the node, add 6 as a pointer element.
- Now, a right-hierarchy of a tree is created, and remaining data values are adjusted accordingly by keeping in mind the applicable rules of equal to or greater than values against the key-value nodes on the right.

Delete Operation

The complexity of the delete procedure in the B+ Tree surpasses that of the insert and search functionality.

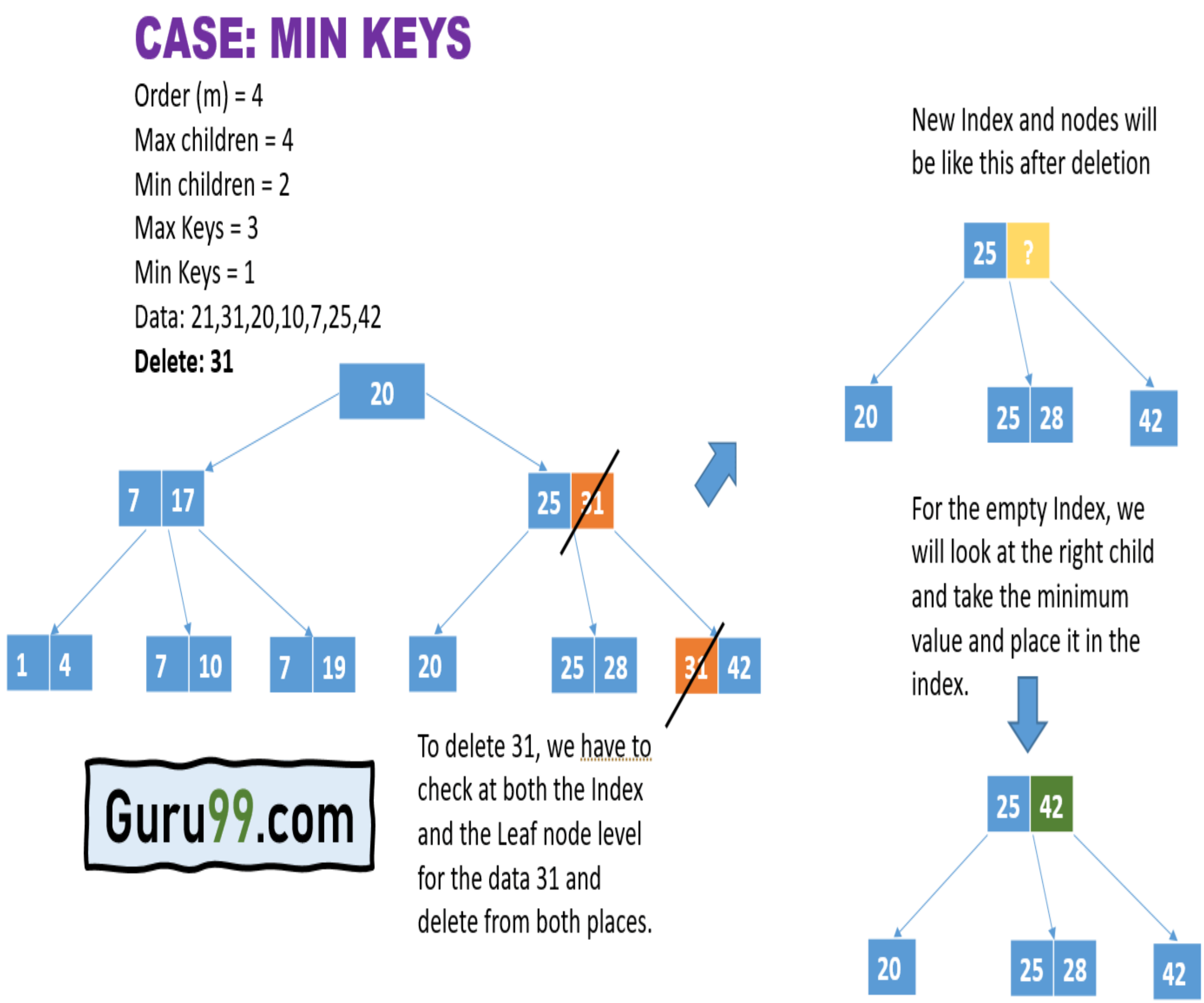
The following algorithm is applicable while deleting an element from the B+ Tree:

- Firstly, we need to locate a leaf entry in the Tree that is holding the key and pointer. , delete the leaf entry from the Tree if the Leaf fulfills the exact conditions of record deletion.
- In case the leaf node only meets the satisfactory factor of being half full, then the operation is completed; otherwise, the Leaf node has minimum entries and cannot be deleted.
- The other linked nodes on the right and left can vacate any entries then move them to the Leaf. If these criteria is not fulfilled, then they should combine the leaf node and its linked node in the tree hierarchy.
- Upon merging of leaf node with its neighbors on the right or left, entries of values in the leaf node or linked neighbor pointing to the top-level node are deleted.



The example above illustrates the procedure to remove an element from the B+ Tree of a specific order.

- Firstly, the exact locations of the element to be deleted are identified in the Tree.
- Here the element to be deleted can only be accurately identified at the leaf level and not at the index placement. Hence, the element can be deleted without affecting the rules of deletion, which is the value of the bare-minimum key.



- In the above example, we have to delete 31 from the Tree.
- We need to locate the instances of 31 in Index and Leaf.
- We can see that 31 is available in both Index and Leaf node level. Hence, we delete it from both instances.

- But, we have to fill the index pointing to 42. We will now look at the right child under 25 and take the minimum value and place it as an index. So, 42 being the only value present, it will become the index.

Delete Operation Algorithm

```
1) Start at the root and go up to leaf node containing the key K
2) Find the node n on the path from the root to the leaf node containing K
    A. If n is root, remove K
         a. if root has more than one key, done
         b. if root has only K
            i) if any of its child nodes can lend a node
               Borrow key from the child and adjust child links
            ii) Otherwise merge the children nodes. It will be a new root
         c. If n is an internal node, remove K
            i) If n has at least ceil(m/2) keys, done!
            ii) If n has less than ceil(m/2) keys,
                If a sibling can lend a key,
                Borrow key from the sibling and adjust keys in n and the parent node
                    Adjust child links
                Else
                    Merge n with its sibling
                    Adjust child links
         d. If n is a leaf node, remove K
            i) If n has at least ceil(M/2) elements, done!
                In case the smallest key is deleted, push up the next key
            ii) If n has less than ceil(m/2) elements
            If the sibling can lend a key
                Borrow key from a sibling and adjust keys in n and its parent node
            Else
                Merge n and its sibling
                Adjust keys in the parent node
```

Output:

The Key "K" is deleted, and keys are borrowed from siblings for adjusting values in n and its parent nodes if needed.

Search Operation

In B+ Tree, a search is one of the easiest procedures to execute and get fast and accurate results from it.

The following search algorithm is applicable:

- To find the required record, you need to execute the binary search on the available records in the Tree.
- In case of an exact match with the search key, the corresponding record is returned to the user.
- In case the exact key is not located by the search in the parent, current, or leaf node, then a "not found message" is displayed to the user.
- The search process can be re-run for better and more accurate results.

Search Operation Algorithm

```
 1. Call the binary search method on the records in the B+ Tree.
 2. If the search parameters match the exact key
            The accurate result is returned and displayed to the user
          Else, if the node being searched is the current and the exact key is not fo
und by the algorithm
            Display the statement "Recordset cannot be found."
```

B+ Tree vs. B Tree

Here, are the main differences between B+ Tree vs. B Tree

B + Tree	B Tree
Search keys can be repeated.	Search keys cannot be redundant.
Data is only saved on the leaf nodes.	Both leaf nodes and internal nodes can store data
Data stored on the leaf node makes the search more accurate and faster.	Searching is slow due to data stored on Leaf and internal nodes.
Deletion is not difficult as an element is only removed from a leaf node.	Deletion of elements is a complicated and time-consuming process.
Linked leaf nodes make the search efficient and quick.	You cannot link leaf nodes.

Problems:

1. Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10 here m=3 d=1