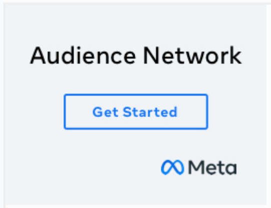


(https://srv.carbonads.net/ads/click/x/GTND42JW
segment=placement:wwwprogramizcom;)
ADS VIA CARBON
(HTTP://CARBONADS.NET/?
CE=WWWPROGRAMIZCOM&UTM_MEDIUM=AD_VIA_LINK&UTM_CAMPAIGN=IN_UNIT&UTM_TERM=CARBON)

Deletion from a B+ Tree

In this tutorial, you will learn about deletion operation on a B+ tree. Also, you will find working examples of deleting elements from a B+ tree in C, C++, Java and Python.

Deleting an element on a B+ tree consists of three main events: **searching** the node where the key to be deleted exists, deleting the key and balancing the tree if required.**Underflow** is a situation when there is less number of keys in a node than the minimum number of keys it should hold.

Deletion Operation

Before going through the steps below, one must know these facts about a B+ tree of degree **m**.

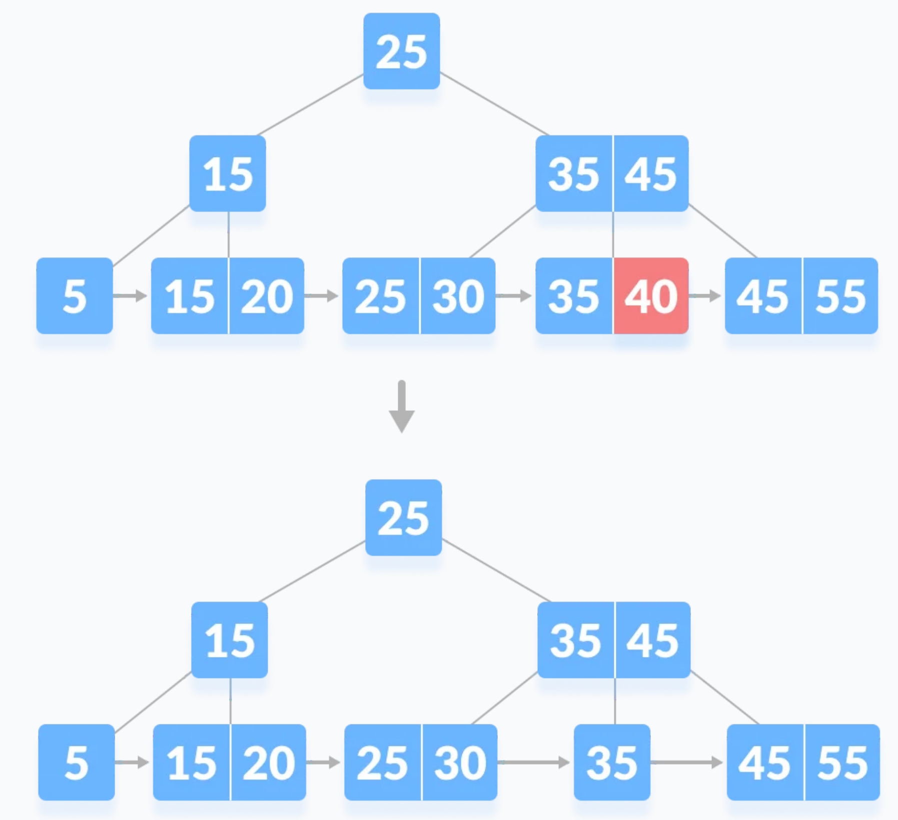
1. A node can have a maximum of m children. (i.e. 3)
2. A node can contain a maximum of `m - 1` keys. (i.e. 2)
3. A node should have a minimum of `⌈m/2⌉` children. (i.e. 2)
4. A node (except root node) should contain a minimum of `⌈m/2⌉ - 1` keys. (i.e. 1)

While deleting a key, we have to take care of the keys present in the internal nodes (i.e. indexes) as well because the values are redundant in a B+ tree. Search the key to be deleted then follow the following steps.

Case I

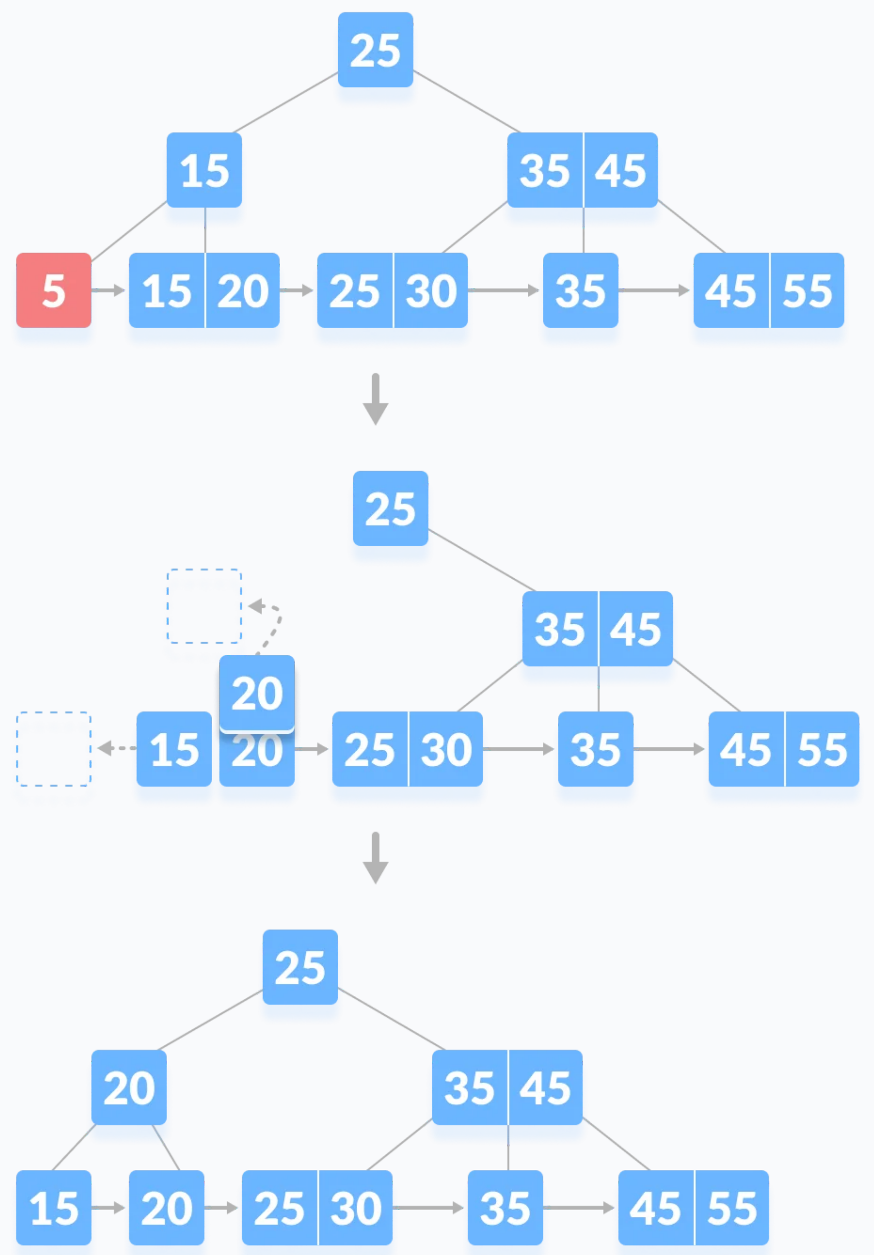
The key to be deleted is present only at the leaf node not in the indexes (or internal nodes). There are two cases for it:

1. There is more than the minimum number of keys in the node. Simply delete the key.



Deleting 40 from B-tree

2. There is an exact minimum number of keys in the node. Delete the key and borrow a key from the immediate sibling. Add the median key of the sibling node to the parent.

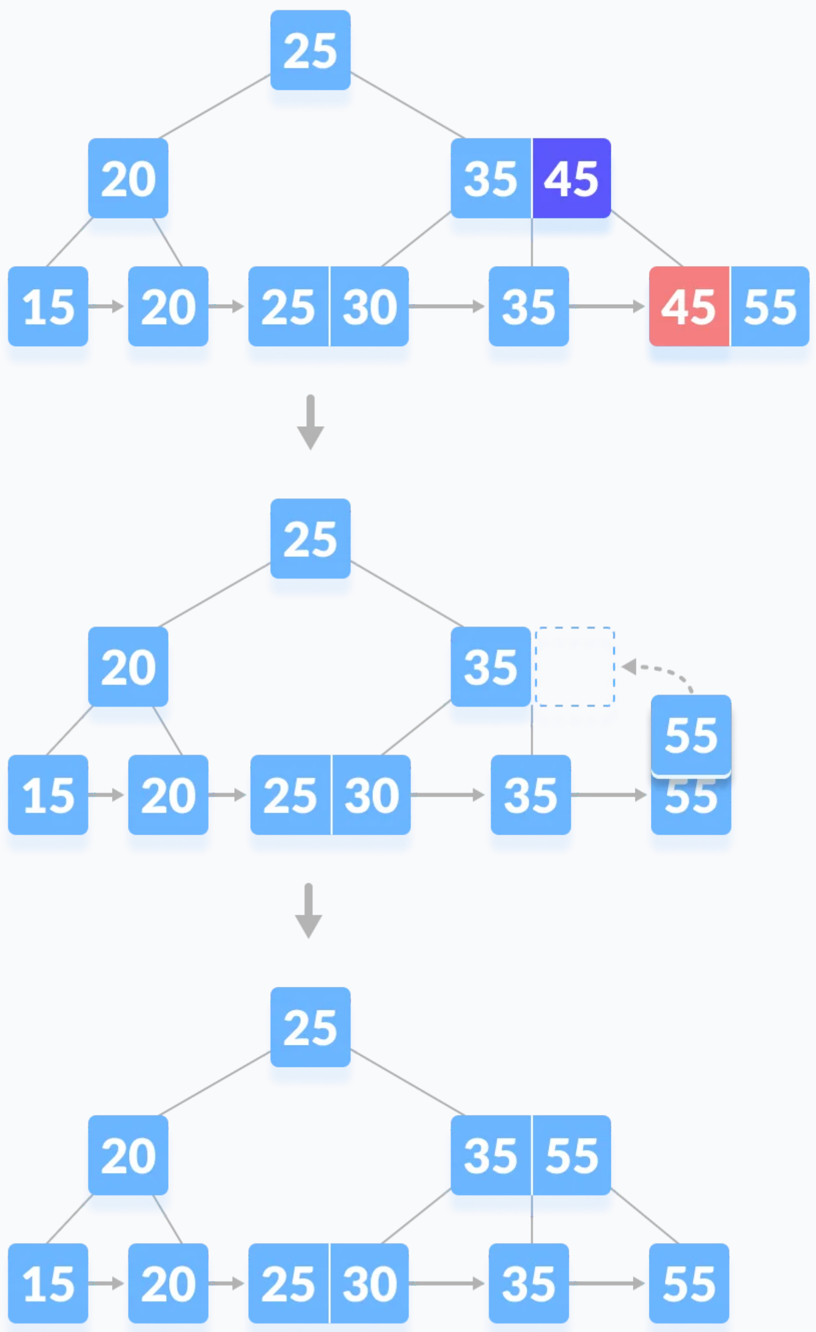


Deleting 5 from B-tree

Case II

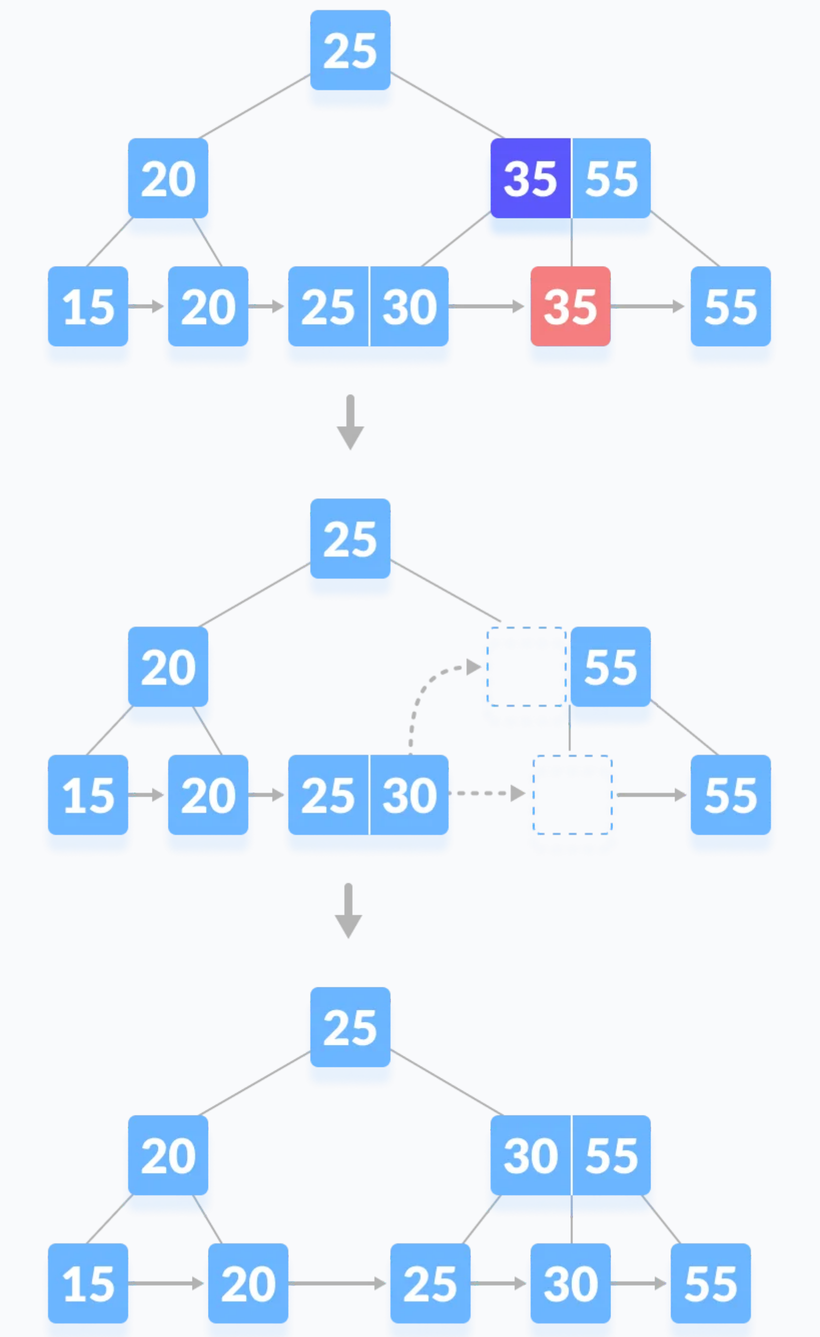
The key to be deleted is present in the internal nodes as well. Then we have to remove them from the internal nodes as well. There are the following cases for this situation.

1. If there is more than the minimum number of keys in the node, simply delete the key from the leaf node and delete the key from the internal node as well.
   Fill the empty space in the internal node with the inorder successor.



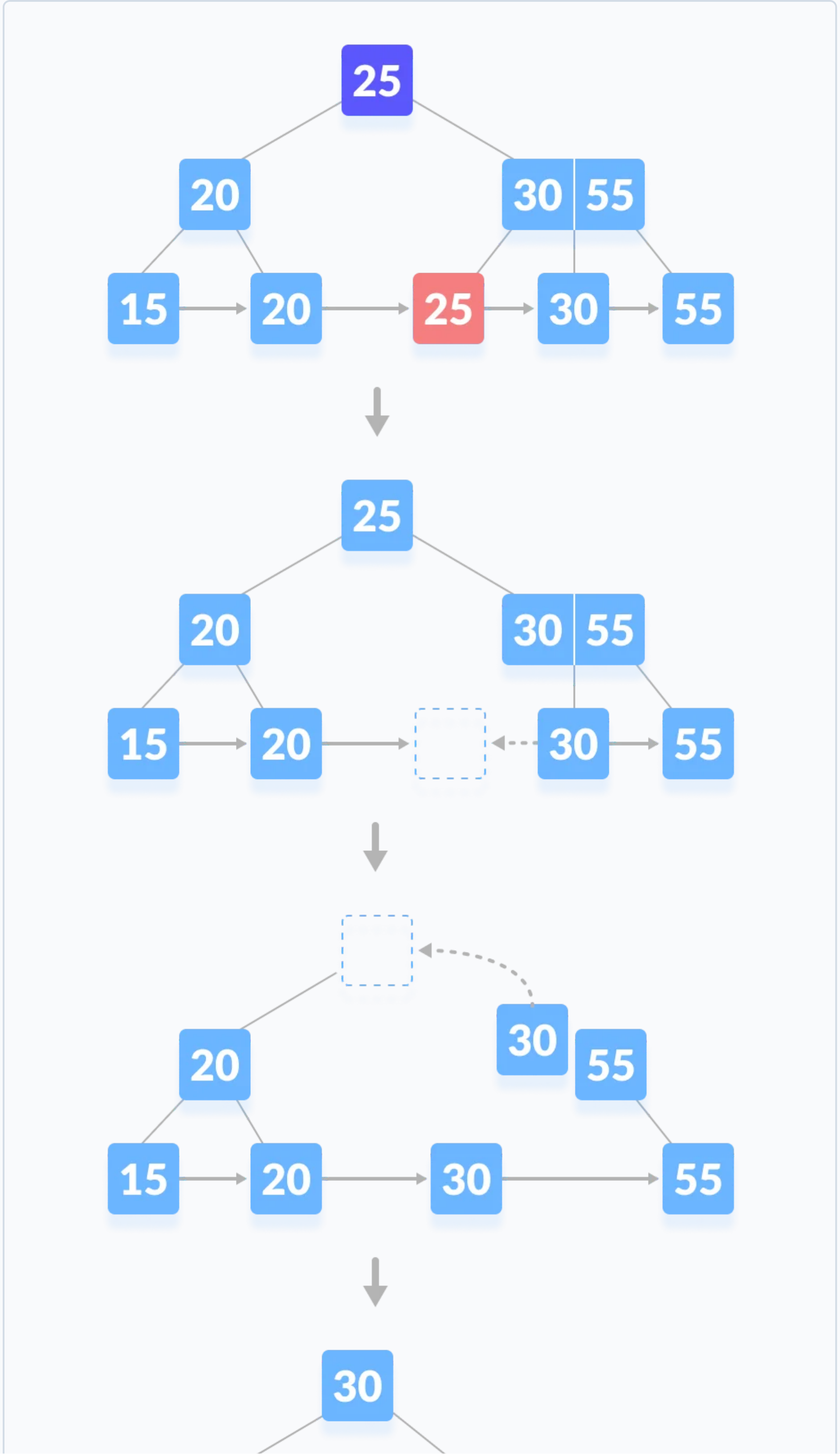
Deleting 45 from B-tree

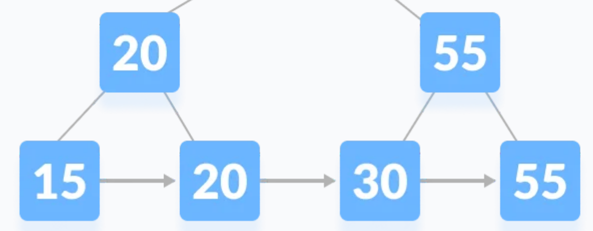
2. If there is an exact minimum number of keys in the node, then delete the key and borrow a key from its immediate sibling (through the parent).
   Fill the empty space created in the index (internal node) with the borrowed key.



Deleting 35 from B-tree

3. This case is similar to Case II(1) but here, empty space is generated above the immediate parent node.
   After deleting the key, merge the empty space with its sibling.
   Fill the empty space in the grandparent node with the inorder successor.

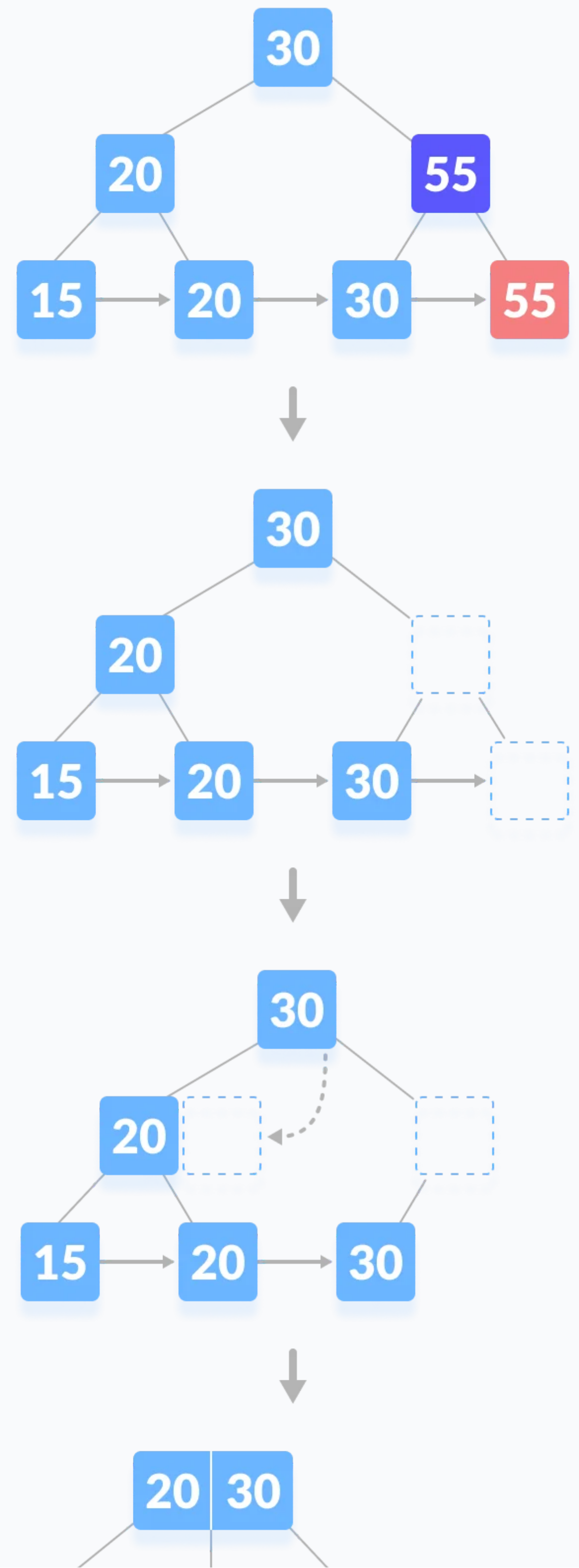

25
20
30 55
15
20
25
30
55
25
20
30 55
15
20
30
55
20
30
55
15
20
30
55
30



Deleting 25 from B-tree

Case III

In this case, the height of the tree gets shrinked. It is a little complicated.Deleting 55 from the tree below leads to this condition. It can be understood in the illustrations below.


30
20
55
15
20
30
55
30
20
15
20
30
30
20
15
20
30
20
30

Deleting 55 from B-tree

Python, Java and C/C++ Examples

Python Java C C++

```
// Deletion operation on a B+ Tree in C++

#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define DEFAULT_ORDER 3

typedef struct record {
  int value;
} record;

typedef struct node {
  void **pointers;
  int *keys;
  struct node *parent;
  bool is_leaf;
  int num_keys;
  struct node *next;
} node;

int order = DEFAULT_ORDER;
node *queue = NULL;
bool verbose_output = false;

void enqueue(node *new_node);
node *dequeue(void);
```

```cpp
// Deletion operation on a B+ tree in C++

#include <climits>
#include <fstream>
#include <iostream>
#include <sstream>
using namespace std;
int MAX = 3;

class BPTree;
class Node {
  bool IS_LEAF;
  int *key, size;
  Node **ptr;
  friend class BPTree;

   public:
  Node();
};
class BPTree {
  Node *root;
  void insertInternal(int, Node *, Node *);
  void removeInternal(int, Node *, Node *);
  Node *findParent(Node *, Node *);

   public:
  BPTree();
```

Deletion Complexity

Time complexity: $\Theta(t.\log_t n)$

The complexity is dominated by $\Theta(\log_t n)$.

Next Tutorial:
Red-Black Tree
(/dsa/red-black-tree)

Previous Tutorial:
Insertion on a B+ tree
(/dsa/insertion-on-a-b-plus-tree)

Share on:

(https://www.facebook.com/sharer/sharer.php?u=https://www.programiz.com/dsa/deletion-from-a-b-plus-tree)

(https://twitter.com/intent/tweet?text=Check%20this%20amazing%20
from-a-b-plus-tree)

Did you find this article helpful?

😃 🙁

Ad

Related Tutorials

DS & Algorithms

Insertion on a B+ Tree

(/dsa/insertion-on-a-b-plus-tree)

DS & Algorithms

B+ Tree

(/dsa/b-plus-tree)

DS & Algorithms

Insertion into a B-tree

(/dsa/insertion-into-a-b-tree)

DS & Algorithms

Perfect Binary Tree

(/dsa/perfect-binary-tree)