B+ Tree

A B+ tree is an advanced form of a self-balancing tree in which all the values are present in the leaf level.

An important concept to be understood before learning B+ tree is multilevel indexing. In multilevel indexing, the index of indices is created as in figure below. It makes accessing the data easier and faster

Properties of a B+ Tree

1. All leaves are at the same level.
2. The root has at least two children.
3. Each node except root can have a maximum of `m` children and at least `m/2` children.
4. Each node can contain a maximum of `m` `- 1` keys and a minimum of `⌈m/2⌉` `-` `1` keys.

Searching on a B+ Tree

The following steps are followed to search for data in a B+ Tree of order `m`. Let the data to be searched be `k`.

1. Start from the root node. Compare k with the keys at the root node `[k1, k2, k3,......km - 1]`.
2. If `k < k1`, go to the left child of the root node.
3. Else if `k == k1`, compare `k2`. If `k < k2`, k lies between `k1` and `k2`. So, search in the left child of `k2`.
4. If `k > k2`, go for `k3, k4,...km-1` as in steps 2 and 3.
5. Repeat the above steps until a leaf node is reached.
6. If k exists in the leaf node, return true else return false.

Insertion on a B+ Tree

Inserting an element into a B+ tree consists of three main events: **searching the appropriate leaf**, **inserting** the element and **balancing/splitting** the tree.

Insertion Operation

Before inserting an element into a B+ tree, these properties must be kept in mind.

- The root has at least two children.
- Each node except root can have a maximum of `m` children and at least `m/2` children.
- Each node can contain a maximum of `m - 1` keys and a minimum of `⌈m/2⌉ - 1` keys.

The following steps are followed for inserting an element.

1. Since every element is inserted into the leaf node, go to the appropriate leaf node.
2. Insert the key into the leaf node.

Case I

1. If the leaf is not full, insert the key into the leaf node in increasing order.

Case II

1. If the leaf is full, insert the key into the leaf node in increasing order and balance the tree in the following way.
2. Break the node at `m/2th` position.
3. Add `m/2th` key to the parent node as well.
4. If the parent node is already full, follow steps 2 to 3.

Deletion from a B+ Tree

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**.

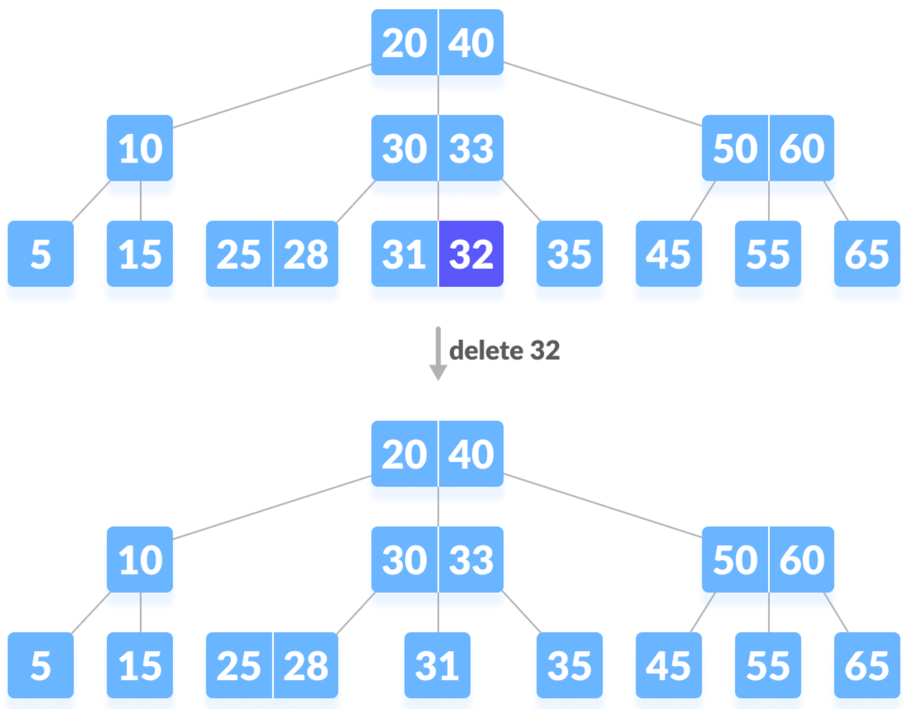
1. A node can have a maximum of m children. (i.e. 3)
2. A node can contain a maximum of `m - 1` keys. (i.e. 2)
3. A node should have a minimum of `⌈m/2⌉` children. (i.e. 2)
4. A node (except root node) should contain a minimum of `⌈m/2⌉ - 1` keys. (i.e. 1)

Case I

The key to be deleted lies in the leaf. There are two cases for it.

The deletion of the key does not violate the property of the minimum number of keys a node should hold.

In the tree below, deleting 32 does not violate the above properties.

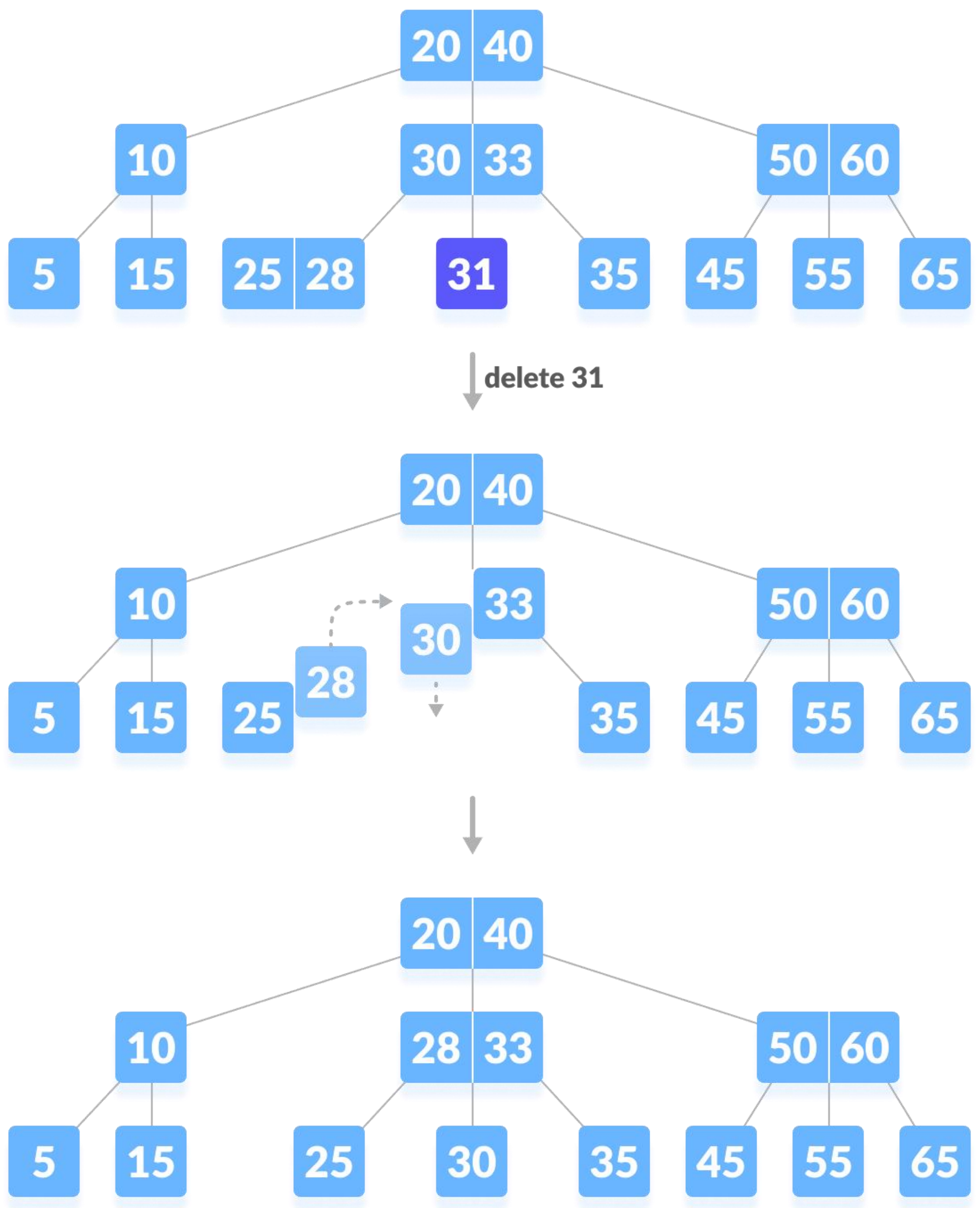


2. The deletion of the key violates the property of the minimum number of keys a node should hold. In this case, we borrow a key from its immediate neighboring sibling node in the order of left to right.

   First, visit the immediate left sibling. If the left sibling node has more than a minimum number of keys, then borrow a key from this node.

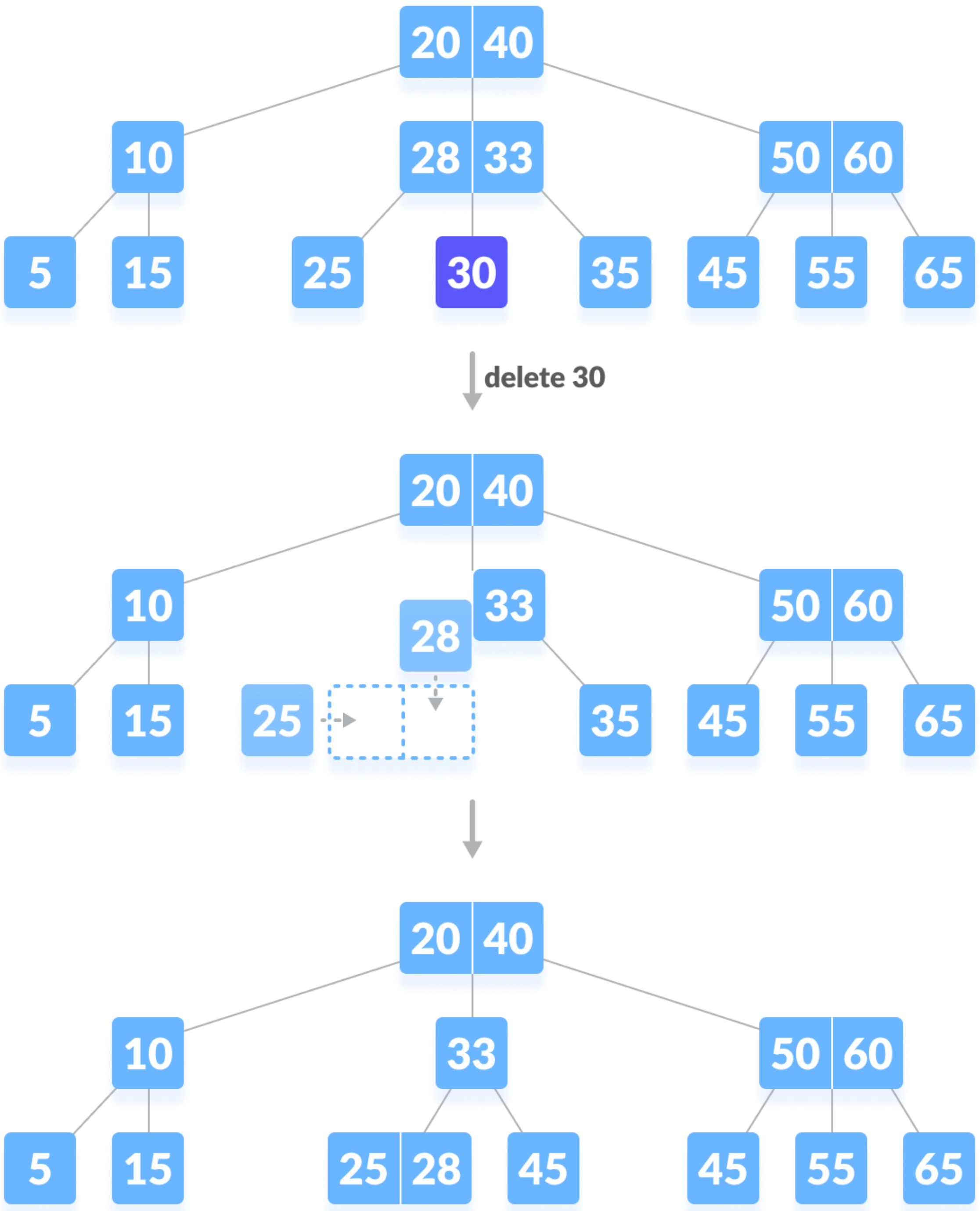
   Else, check to borrow from the immediate right sibling node.

   In the tree below, deleting 31 results in the above condition. Let us borrow a key from the left sibling node.


20 40
10
30 33
50 60
5
15
25 28
31
35
45
55
65
delete 31
20 40
10
33
30
28
50 60
5
15
25
35
45
55
65
20 40
10
28 33
50 60
5
15
25
30
35
45
55
65

If both the immediate sibling nodes already have a minimum number of keys, then merge the node with either the left sibling node or the right sibling node. **This merging is done through the parent node.**

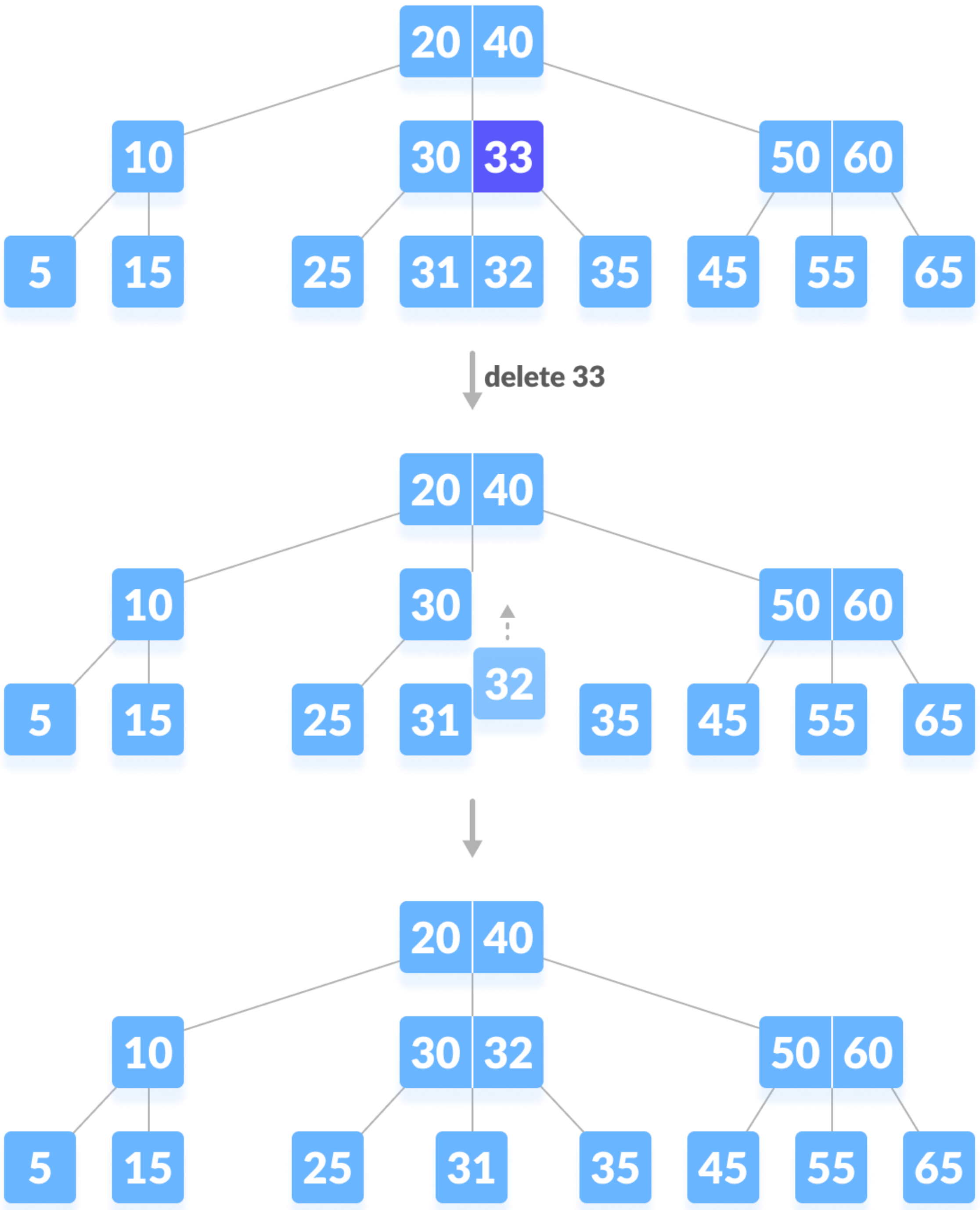
Deleting 30 results in the above case.



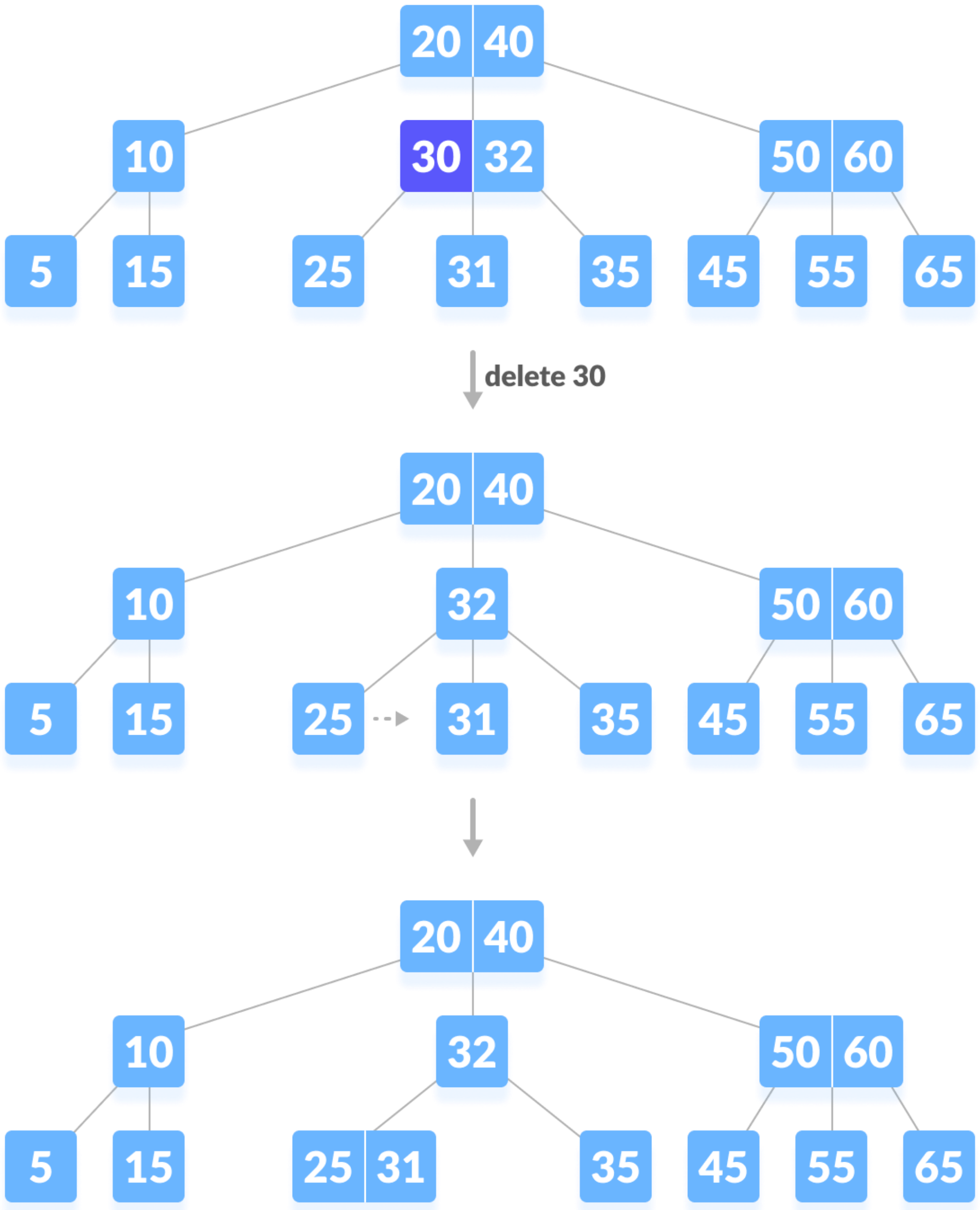
Case II

If the key to be deleted lies in the internal node, the following cases occur.

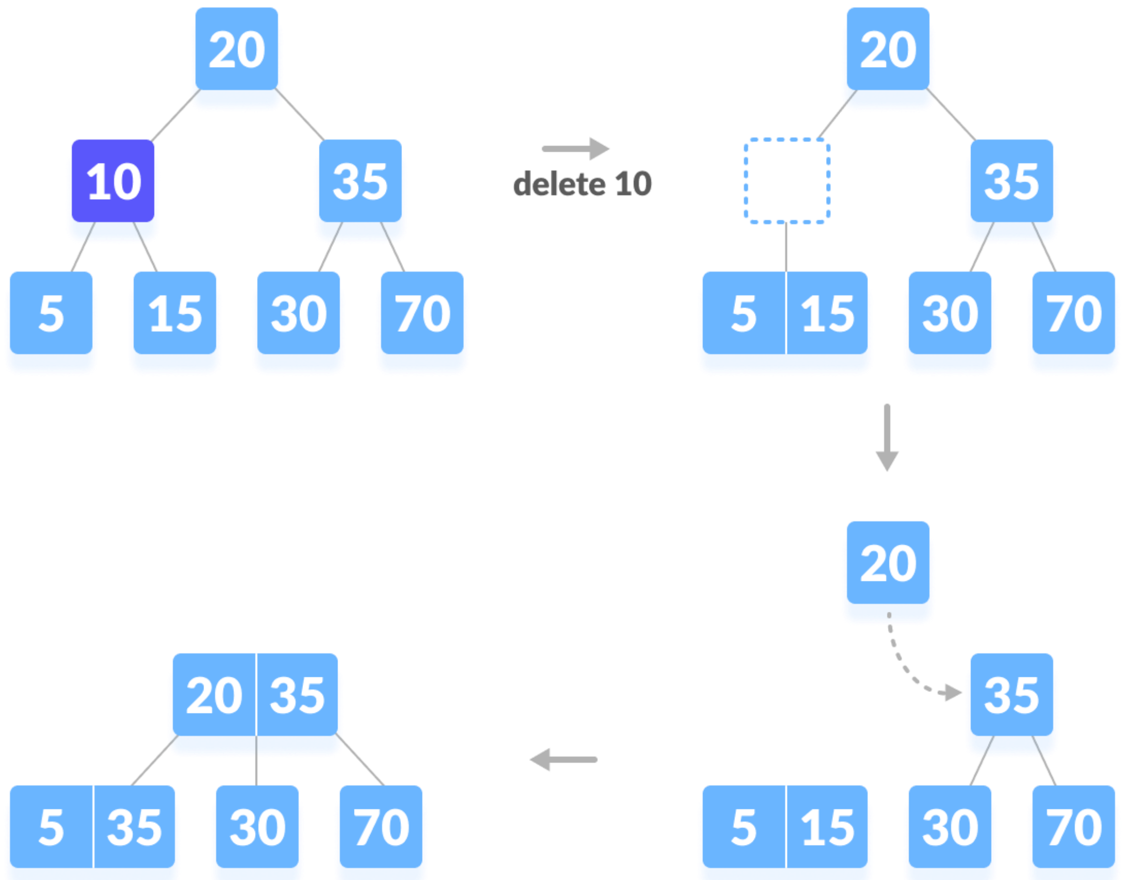
1. The internal node, which is deleted, is replaced by an inorder predecessor if the left child has more than the minimum number of keys.



1. The internal node, which is deleted, is replaced by an inorder successor if the right child has more than the minimum number of keys.
2. If either child has exactly a minimum number of keys then, merge the left and the right children.



Case III



B+ Tree Applications

- Multilevel Indexing
- Faster operations on the tree (insertion, deletion, search)
- Database indexing