Data Structures (15B11CI311)

Odd Semester 2020

3rd Semester , Computer Science and Engineering

Jaypee Institute Of Information Technology (JIIT), Noida

Contents

- B+ tree
- Difference between B-Tree & B+ Tree
- B+ Tree Structure
- Insertion in B+ tree
- Deletion in B+ tree

B+ Tree

- It is a variant of B-trees
- All data of B+ tree is stored at lowest level (leaf nodes)
- All leaves are linked sequentially
- Used as a dynamic indexing method in relational DBs
- Contains index pages and data pages.
- Root and non-leaf nodes are index pages.

Difference between B-Tree & B+ Tree

- B+ trees don’t store data pointer in interior nodes, they are ONLY stored in leaf nodes. This is not optional as in B-Tree. This means that interior nodes can fit more keys on block of memory.
- The leaf nodes of B+ trees are linked, so doing a linear scan of all keys will requires just one pass through all the leaf nodes. A B tree, on the other hand, would require a traversal of every level in the tree. This property can be utilized for efficient search as well since data is stored only in leaves.

B+ Tree Structure

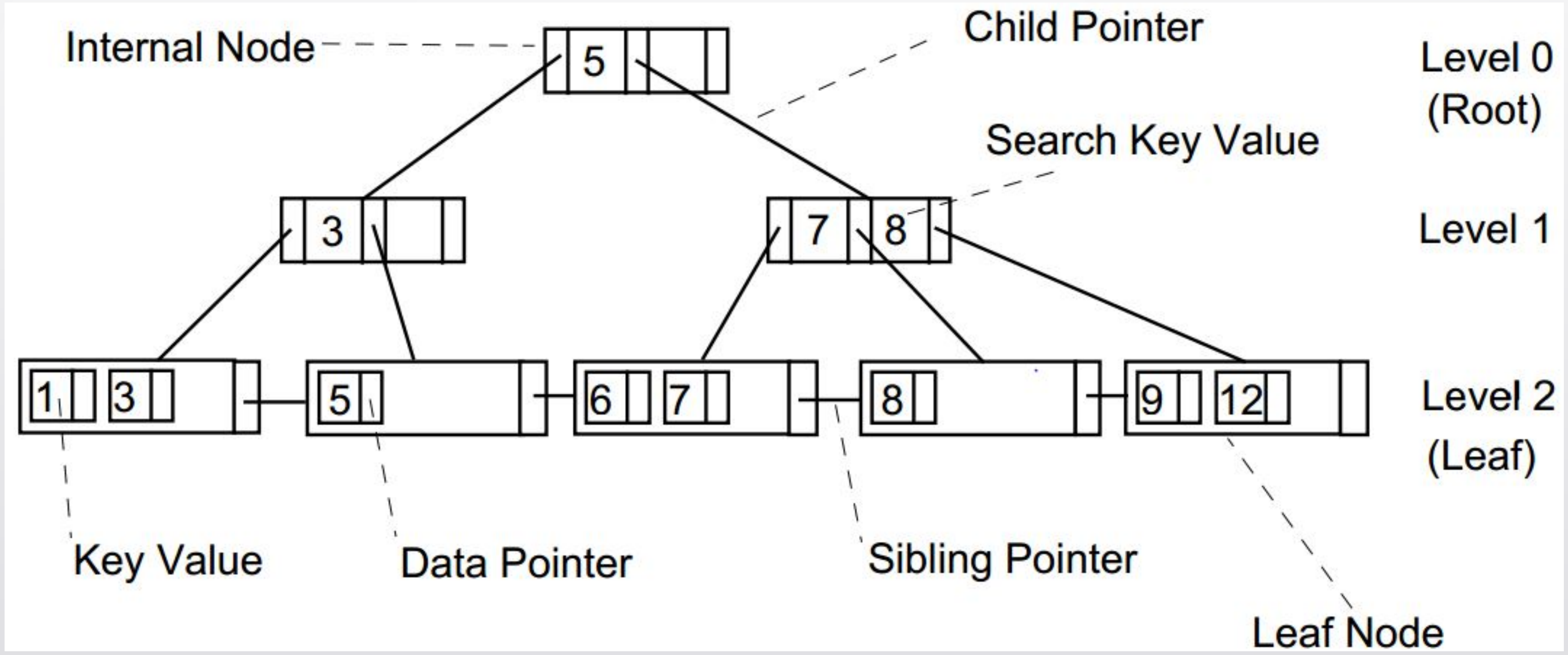
- A B+ tree is a balanced tree where every path from the root of the tree to a leaf of the tree is of the same length.
- „ It contains index pages and data pages.
- „ The internal nodes store key values that serve as place markers to guide the search for a record in a leaf node.
- „ Leaf nodes can store entire records. When a B+ tree is used as an index file they store keys and pointers to records.
- Every node *except root* must be at least ½ full.

B+ Tree Properties

Consider a B+ Tree of degree m:

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

Example of B+ tree



Source: https://iq.opengenus.org/content/images/2018/06/b--search.jpg

B+ Tree Insertion

- The key value determines a record's placement in a B+ Tree. The leaf nodes are maintained in sequential order and a doubly linked list (not shown) connects each leaf page with its sibling page. This doubly linked list speeds the data movement as the pages grow and contract.
- There can be three scenarios when we add a record to a B+ Tree.

B+ Tree Insertion (contd..)

1. If the leaf is not full and index (internal) is not full
   a) Place the record in sorted position in the appropriate leaf node.
2. If the leaf is full and index is not full
   a) Split the leaf node
   b) Place the middle key in the index node in sorted order.
   c) Left leaf node contains records with keys below the middle key
   d) Right leaf node contains records with keys equal to or greater than the middle key.

B+ Tree Insertion (contd..)

3. If the leaf is full and index is full
    a) Split the leaf node
    b) The records with keys < middle key go to the left leaf node.
    c) The records with keys ≥ middle key go to the right leaf node.
    d) Split the index node.
    e) The keys < middle key go to the left index node.
    f) The keys > middle key go to the right index node.
    g) The middle key goes to the next (higher level) index.

If the next level index node is full, continue splitting the index node.

B+ Tree Insertion (contd..)

Note:

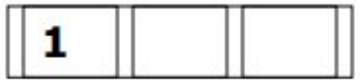
The leaf nodes and the internal nodes are treated differently when they split. When a leaf node splits, a copy of the middle key is moved up to be a separator at the next level. When an internal (index) node splits, the key itself is moved up to act a separator.

Example

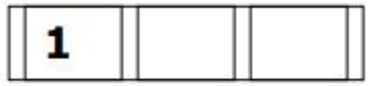
- Suppose each B+ tree node can hold up to 4 pointers and 3 keys.
- Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10
- Show the tree after insertions

Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

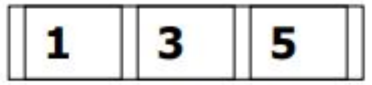
- Insert 1



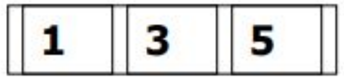
Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10



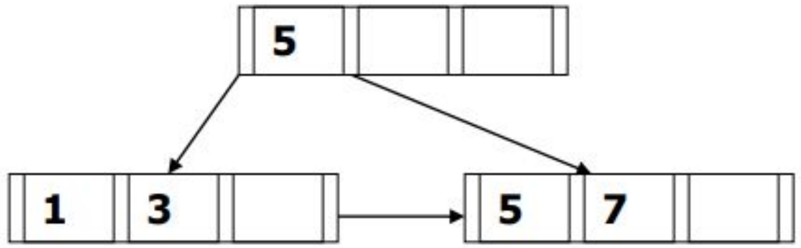
- Insert 3, 5



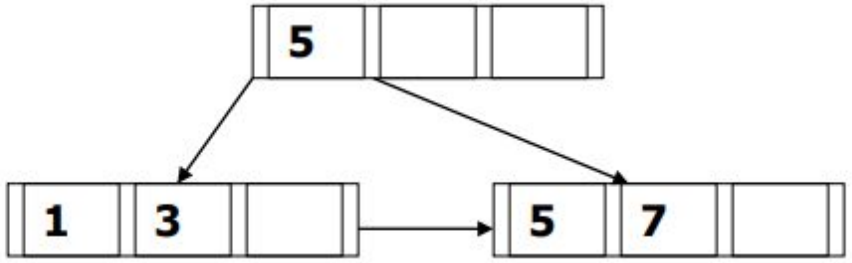
Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10



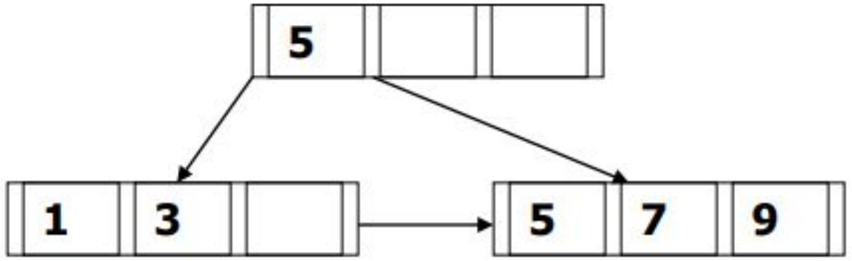
- Insert 7



Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

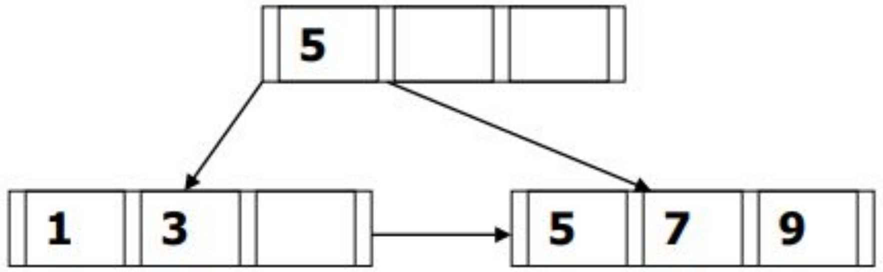


- Insert 9

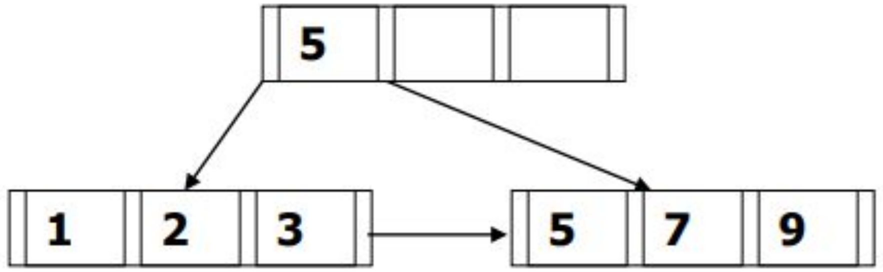


Source: https://www.cs.nmsu.edu/~hcao/teaching/cs582/note/DB2_4_BplusTreeExample.pdf

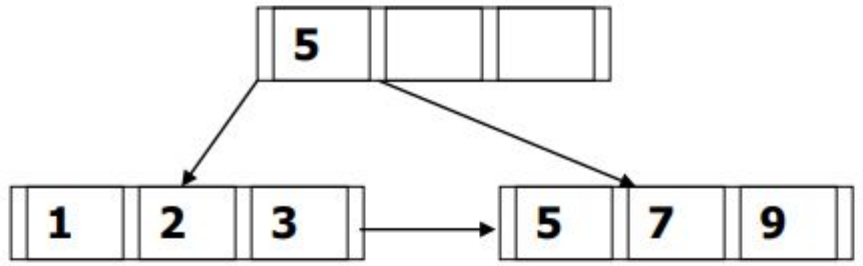
Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10



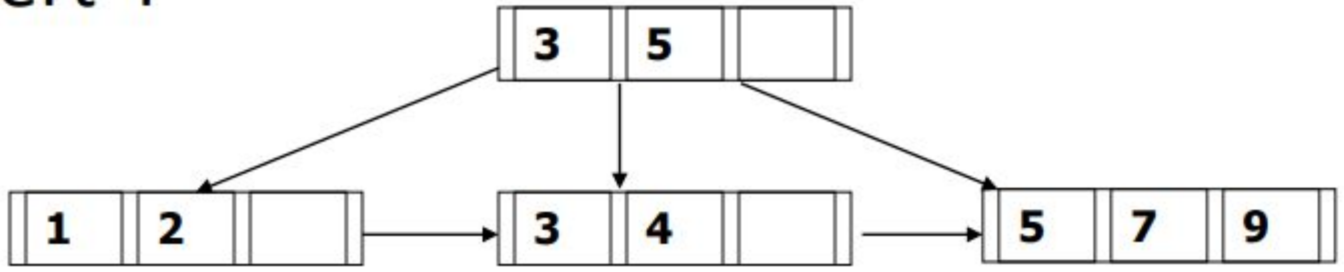
- Insert 2



Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

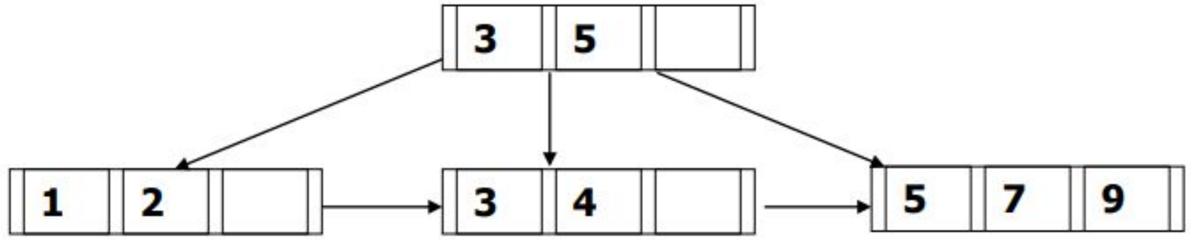


- Insert 4

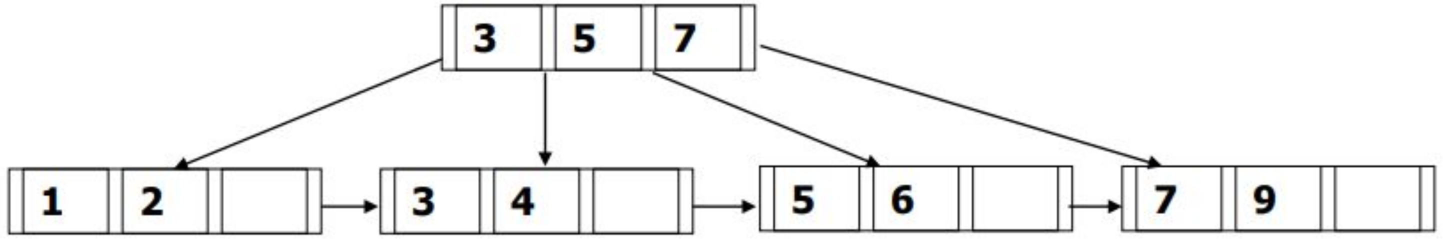


Source: https://www.cs.nmsu.edu/~hcao/teaching/cs582/note/DB2_4_BplusTreeExample.pdf

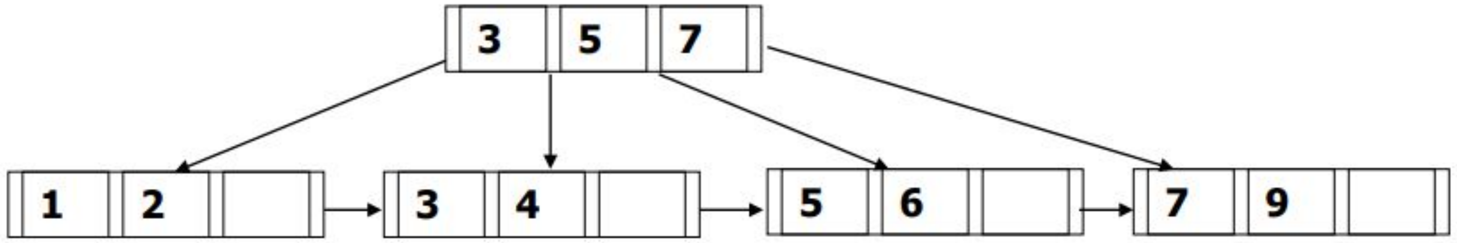
Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10



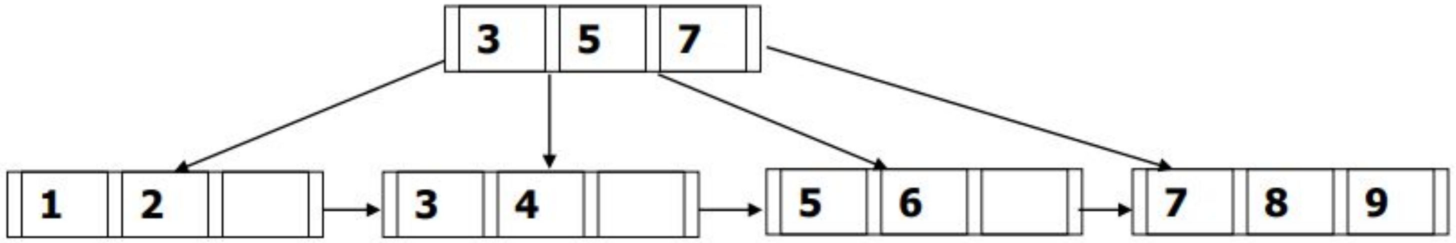
- Insert 6



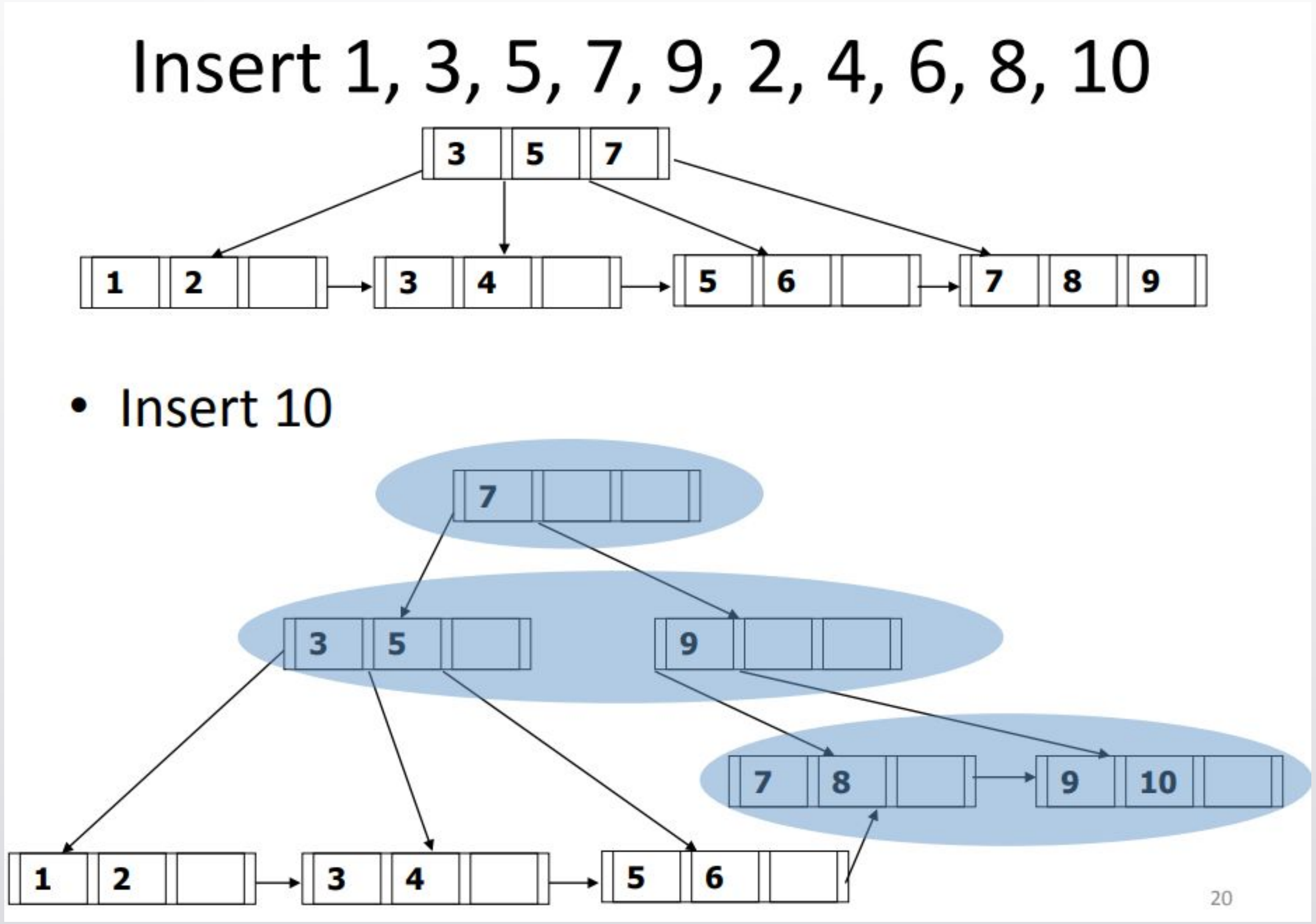
Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10



- Insert 8



Source: https://www.cs.nmsu.edu/~hcao/teaching/cs582/note/DB2_4_BplusTreeExample.pdf



Insert 1, 3, 5, 7, 9, 2, 4, 6, 8, 10

- Insert 10

Note that root was split, leading to increase in height. In this example, we can avoid split by re-distributing entries (by moving the record to its sibling). However, this is usually not done in practice.

Source: https://www.cs.nmsu.edu/~hcao/teaching/cs582/note/DB2_4_BplusTreeExample.pdf

B+ Tree Deletion

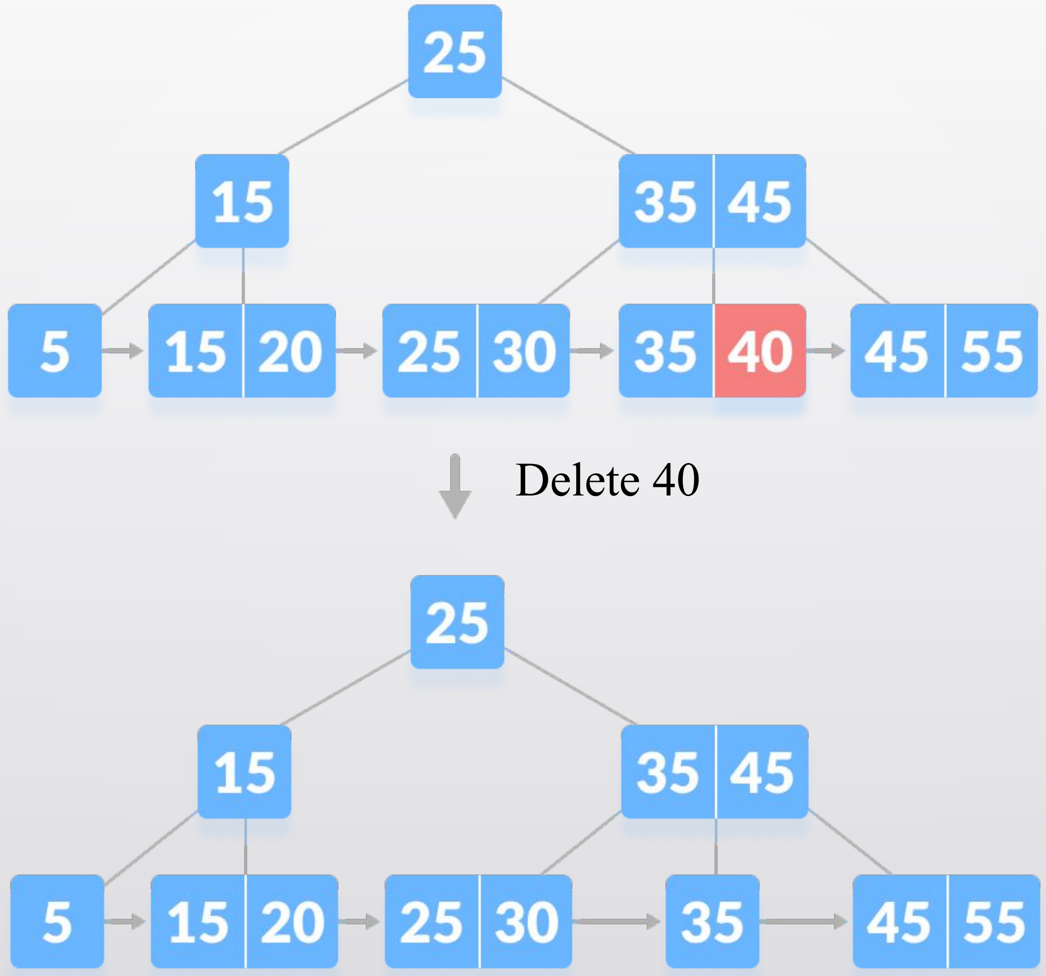
When performing deletion, we also have to take care of the keys present in the index nodes as well because the values may be redundant in the B+ tree. Search the key which is to be deleted then perform the following steps.

Case I

The key to be deleted is present only at the leaf node not in the index node. There are two cases for it:

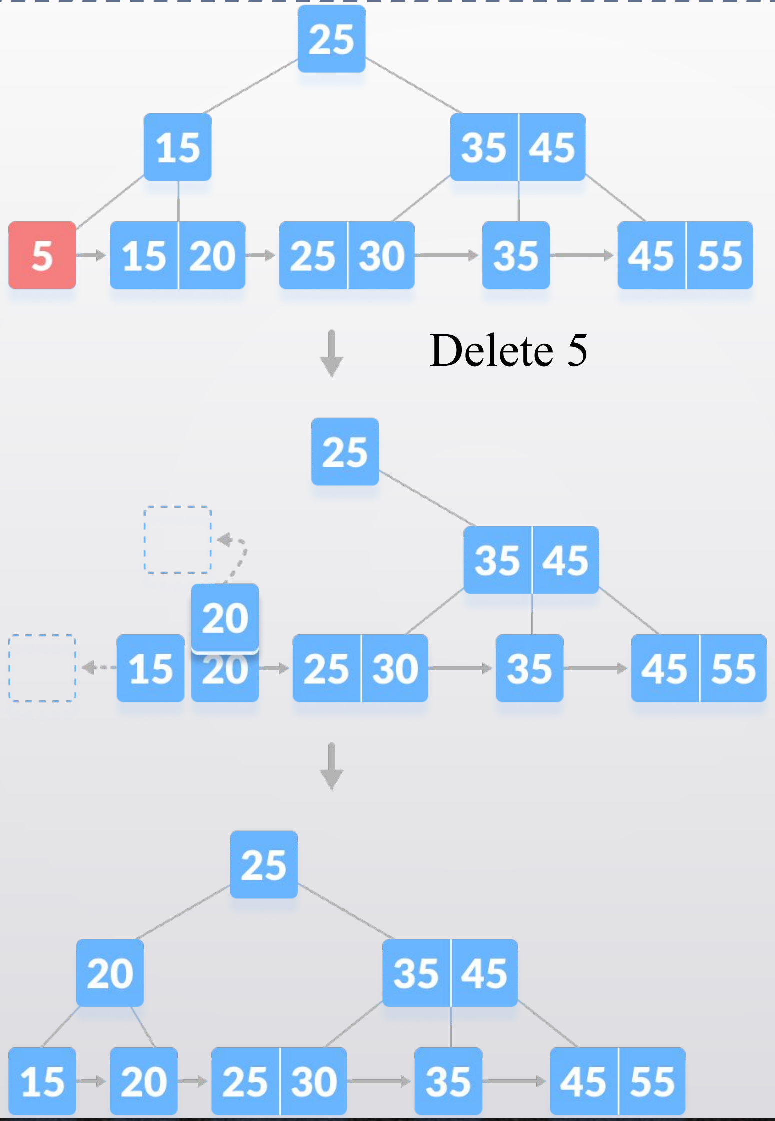
1. There is more than the minimum number of keys in the node. Simply delete the key.
2. There is an exact minimum no. of keys in the node. Delete the key and borrow a key from the immediate sibling. Add the middle key of the sibling node to the parent.

B+ Tree Deletion



Source: https://www.programiz.com/dsa/deletion-from-a-b-plus-tree

B+ Tree Deletion



Source: https://www.programiz.com/dsa/deletion-from-a-b-plus-tree

B+ Tree Deletion

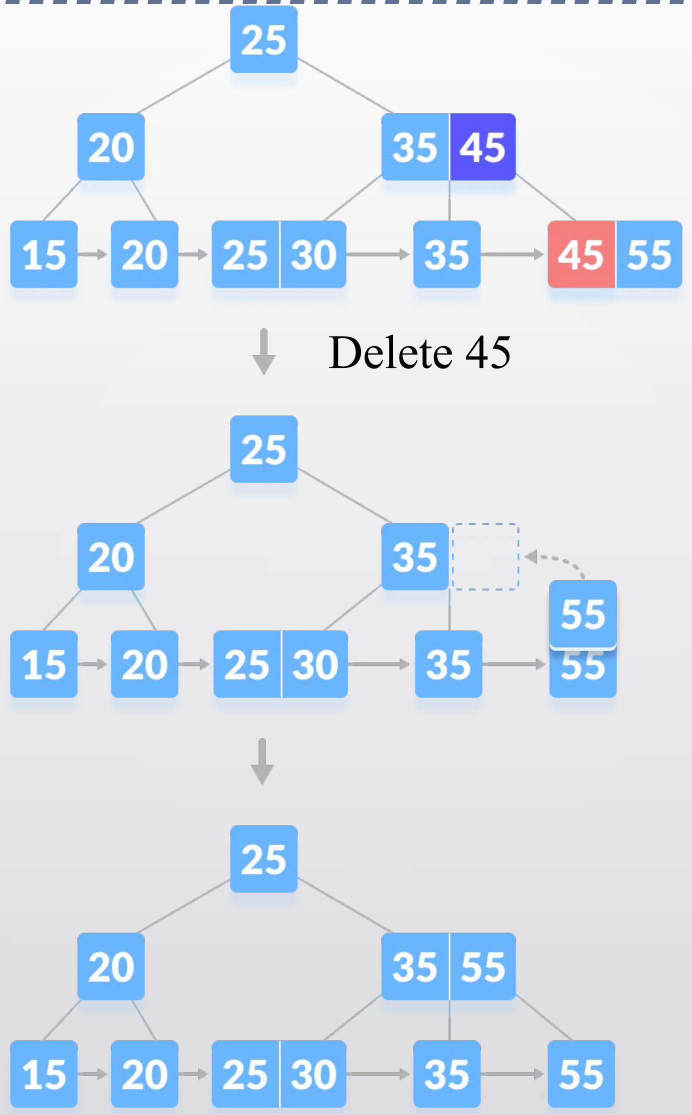
Case II

The key to be deleted is present in the index node as well. Then we have to remove it from the index node as well. There can be the following scenarios for this case:

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 index node as well.
   Fill the empty space in the internal node with the inorder successor.
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.
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.

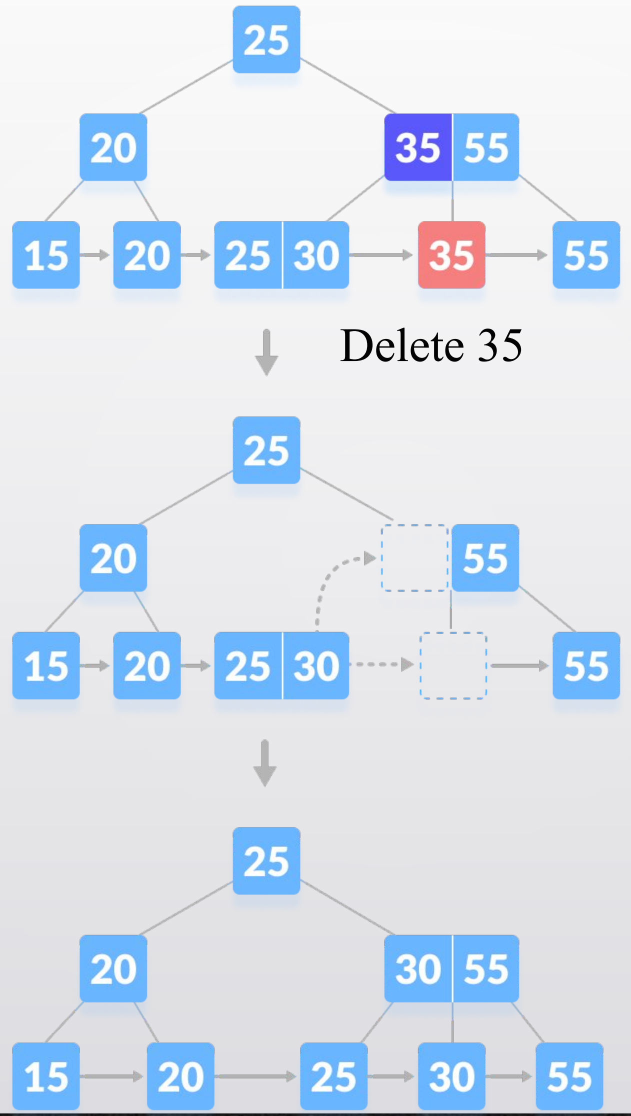
Source: https://www.programiz.com/dsa/deletion-from-a-b-plus-tree

B+ Tree Deletion



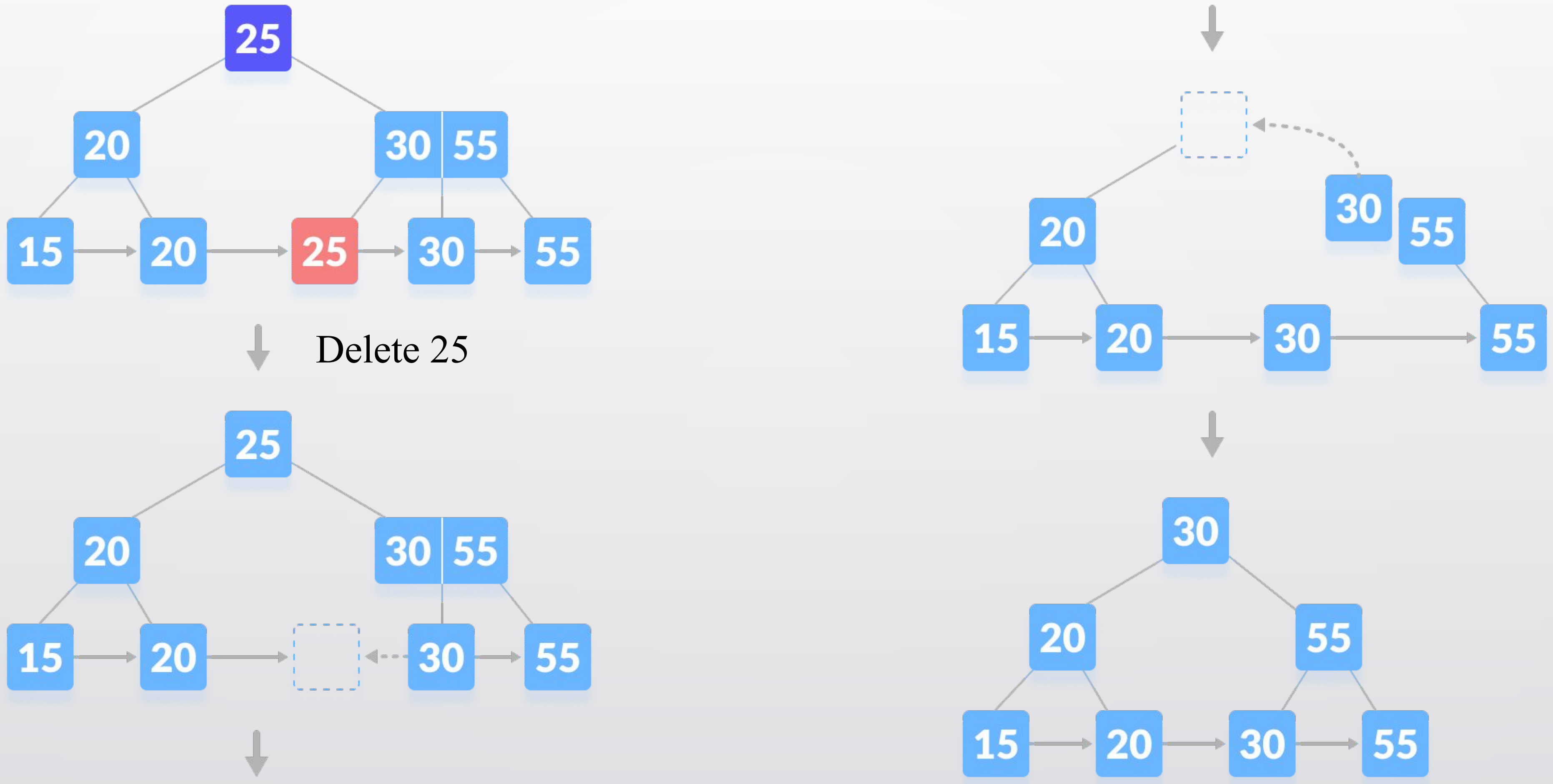
Source: https://www.programiz.com/dsa/deletion-from-a-b-plus-tree

B+ Tree Deletion



Source: https://www.programiz.com/dsa/deletion-from-a-b-plus-tree

B+ Tree Deletion



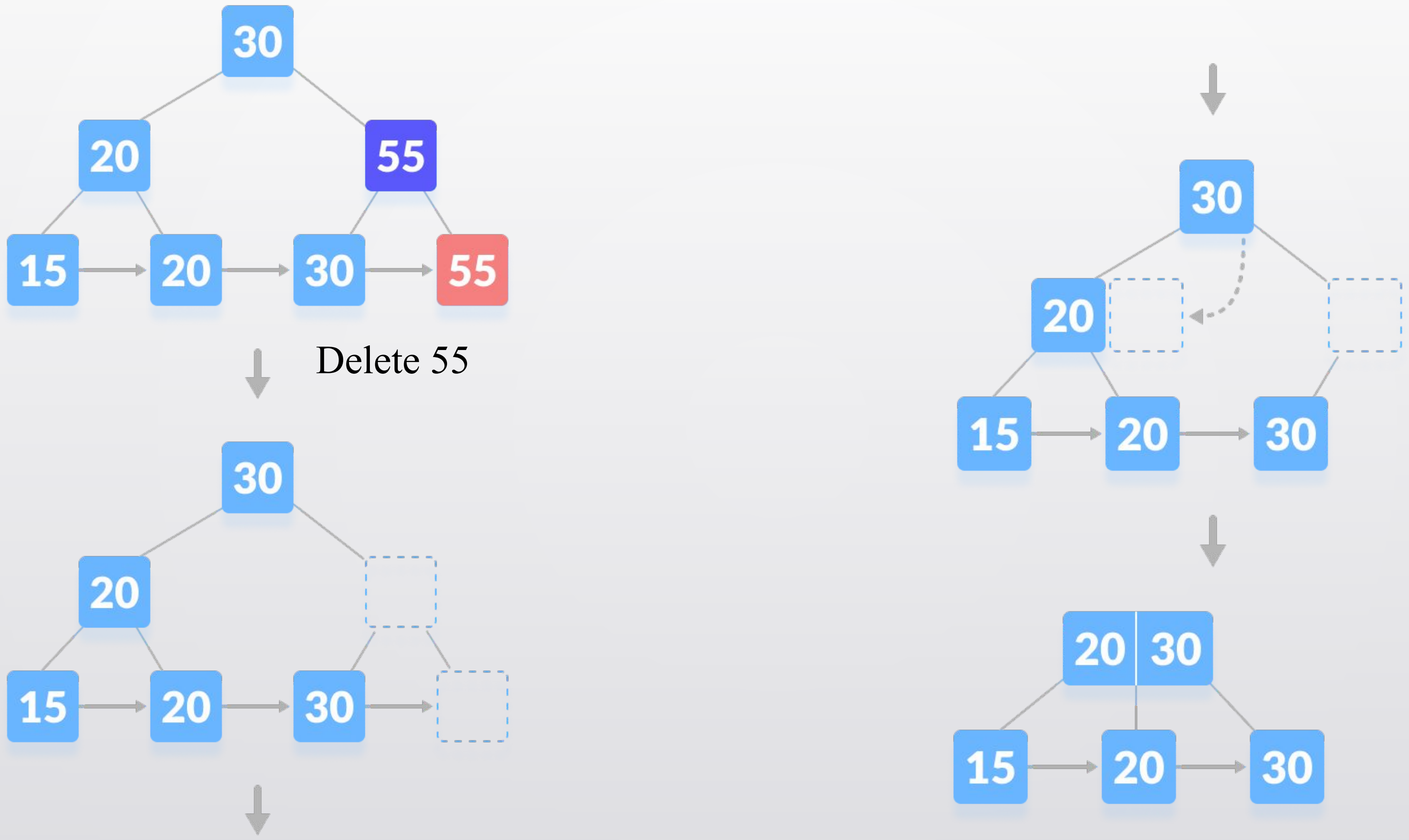
Source: https://www.programiz.com/dsa/deletion-from-a-b-plus-tree

B+ Tree Deletion

Case III

- In this case, the height of the tree will be reduced i.e. the tree is shrinked. It is somewhat complicated.
- When deleting 55 from the tree will lead to this condition.

B+ Tree Deletion



Source: https://www.programiz.com/dsa/deletion-from-a-b-plus-tree

References

- https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html (For Visualization of B+ Trees)
- https://www.cs.nmsu.edu/~hcao/teaching/cs582/note/DB2_4_BplusTreeExample.pdf
- https://www.programiz.com/dsa/deletion-from-a-b-plus-tree