

B+ Tree

	ID#	NAME	ADDRESS	COURSE	LEVEL
Block 1	9701654				
	9701890				

	9702317				
	ID#	NAME	ADDRESS	COURSE	LEVEL
Block 2	9702381				
	9702399				

	9703478				
	ID#	NAME	ADDRESS	COURSE	LEVEL
Block 3	9703501				
	9703569				

	9801220				

	ID#	NAME	ADDRESS	COURSE	LEVEL
Block b-1	9902318				
	9902449				

	9903778				
	ID#	NAME	ADDRESS	COURSE	LEVEL
Block b	9903791				
	9903799				

	9903988				

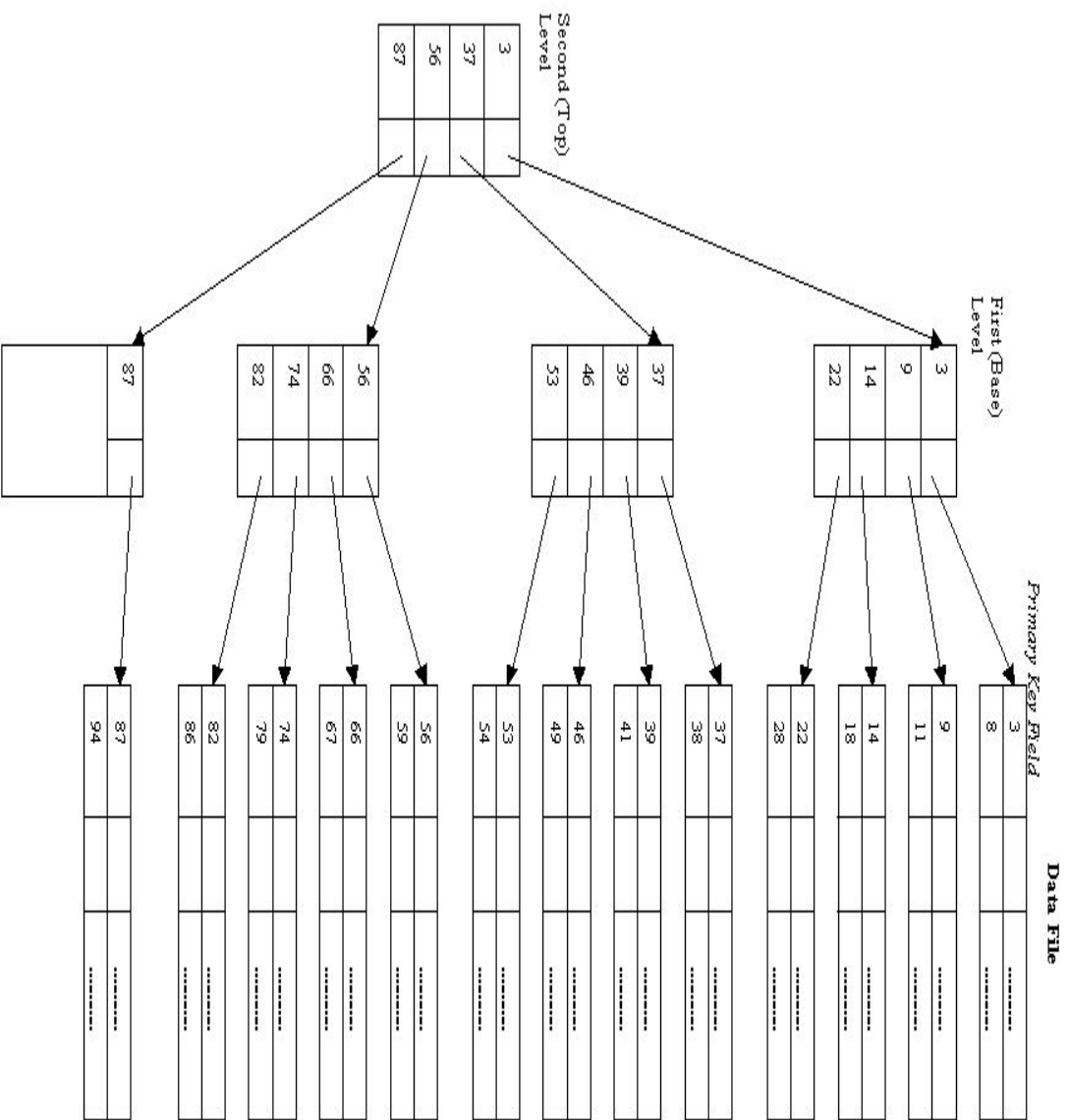
Index File

Block Anchor
Primary Key
Value

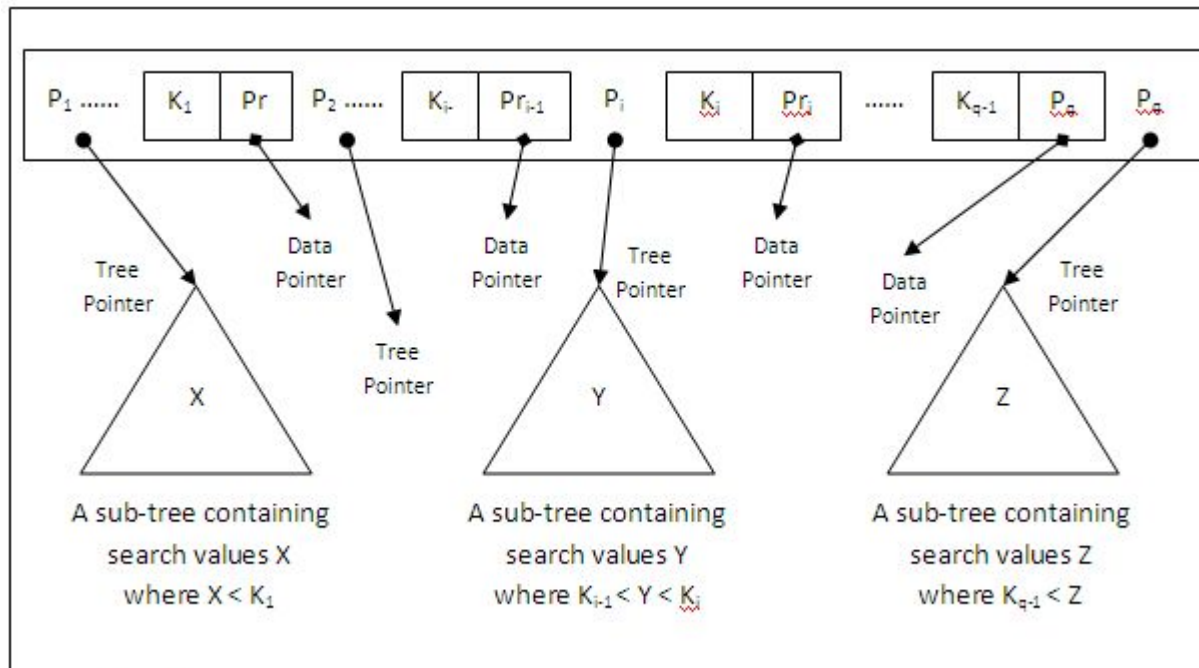
Block
Pointer

9701654	
9702381	
9703501	
9902318	
9903791	

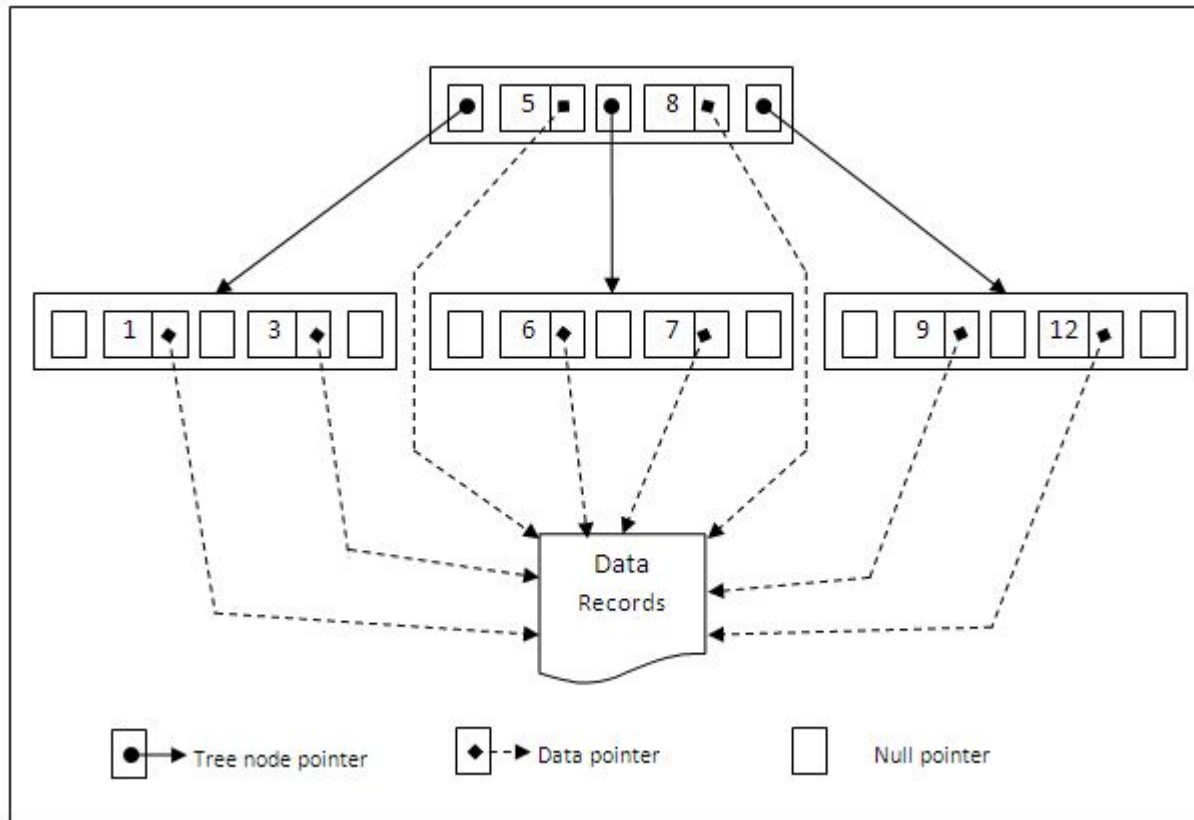
	ID#	NAME	ADDRESS	COURSE	LEVEL
Block 1	9701654				
	9701890				
				
	9702317				
	ID#	NAME	ADDRESS	COURSE	LEVEL
Block 2	9702381				
	9702399				
				
	9703478				
	ID#	NAME	ADDRESS	COURSE	LEVEL
Block 3	9703501				
	9703569				
				
	9801220				
				
	ID#	NAME	ADDRESS	COURSE	LEVEL
Block b-1	9902318				
	9902449				
				
	9903778				
	ID#	NAME	ADDRESS	COURSE	LEVEL
Block b	9903791				
	9903799				
				
	9903988				



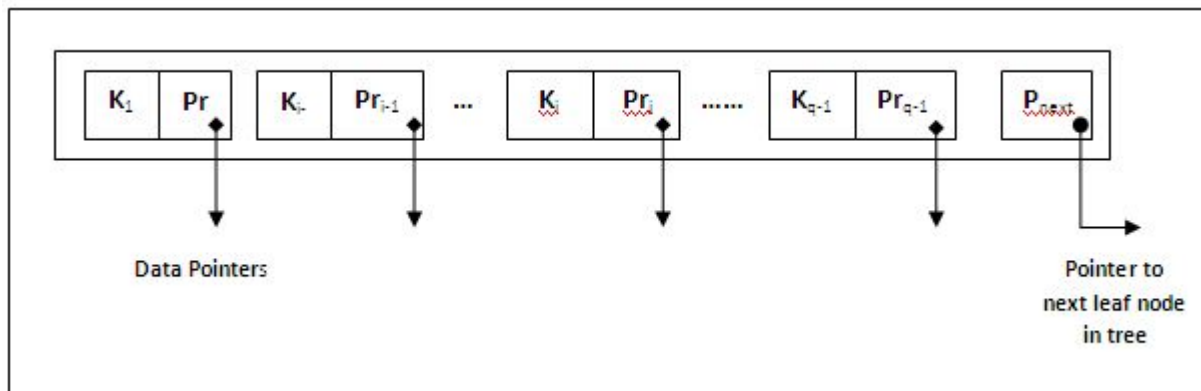
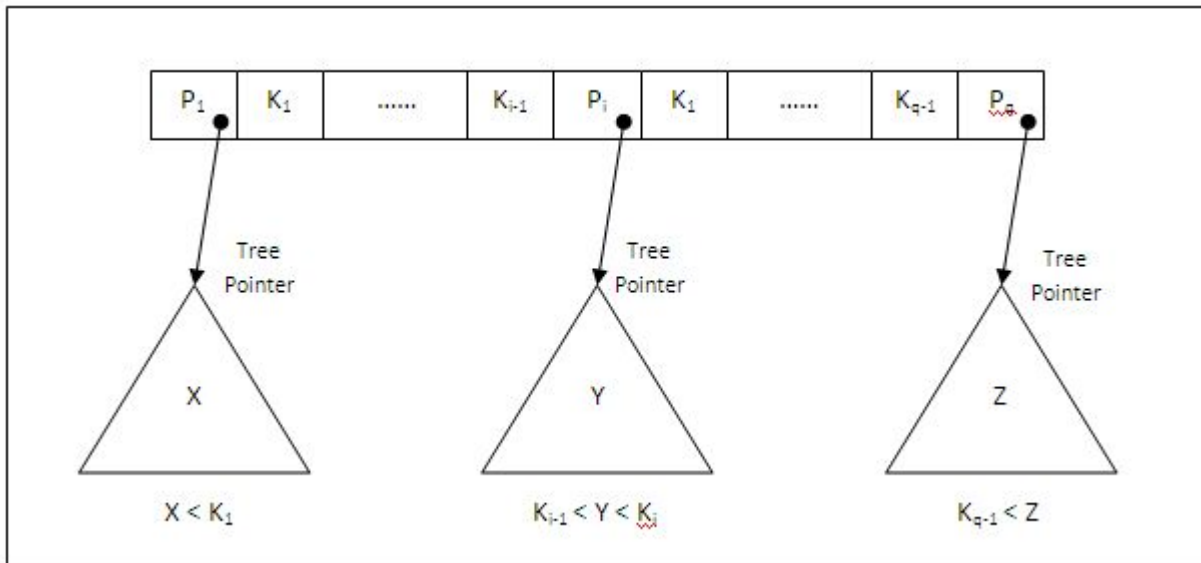
B-Tree

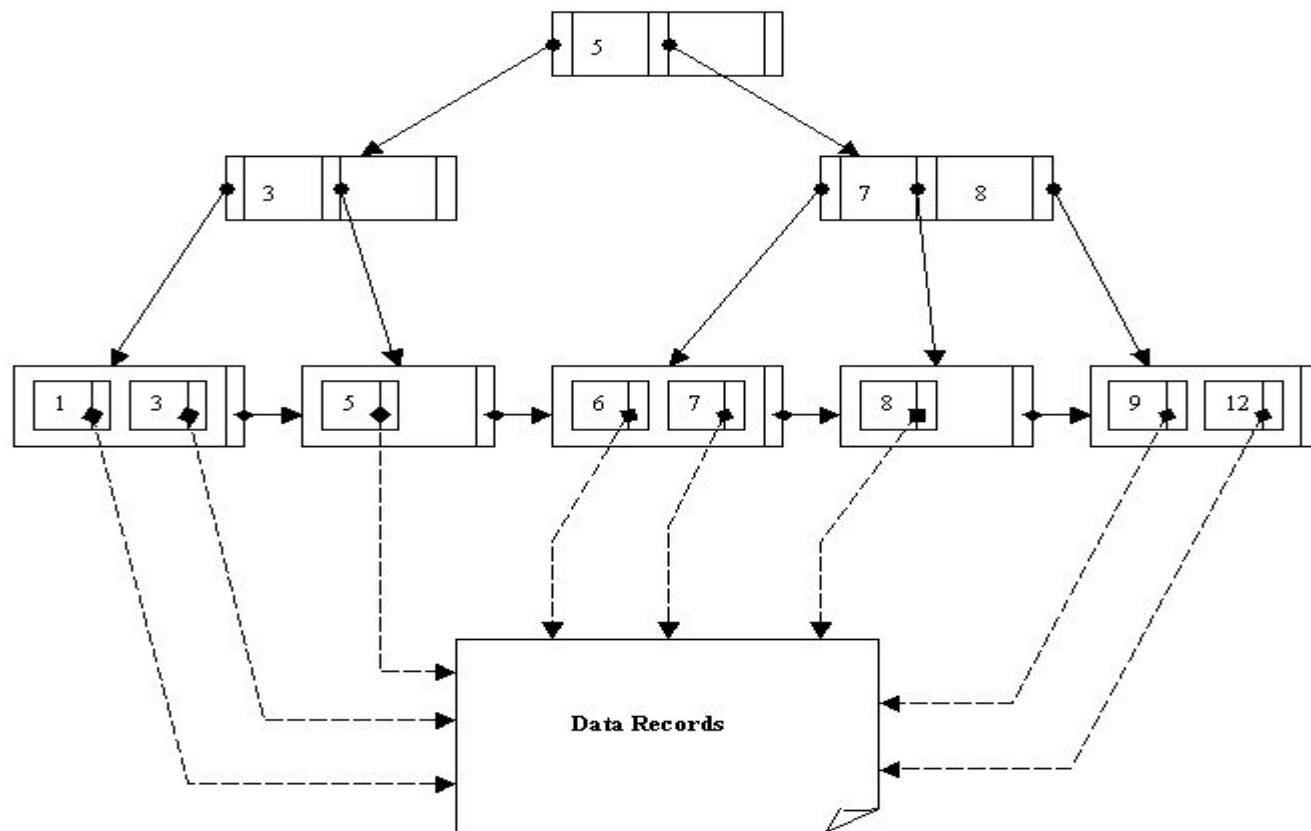



B-tree




B+ Tree



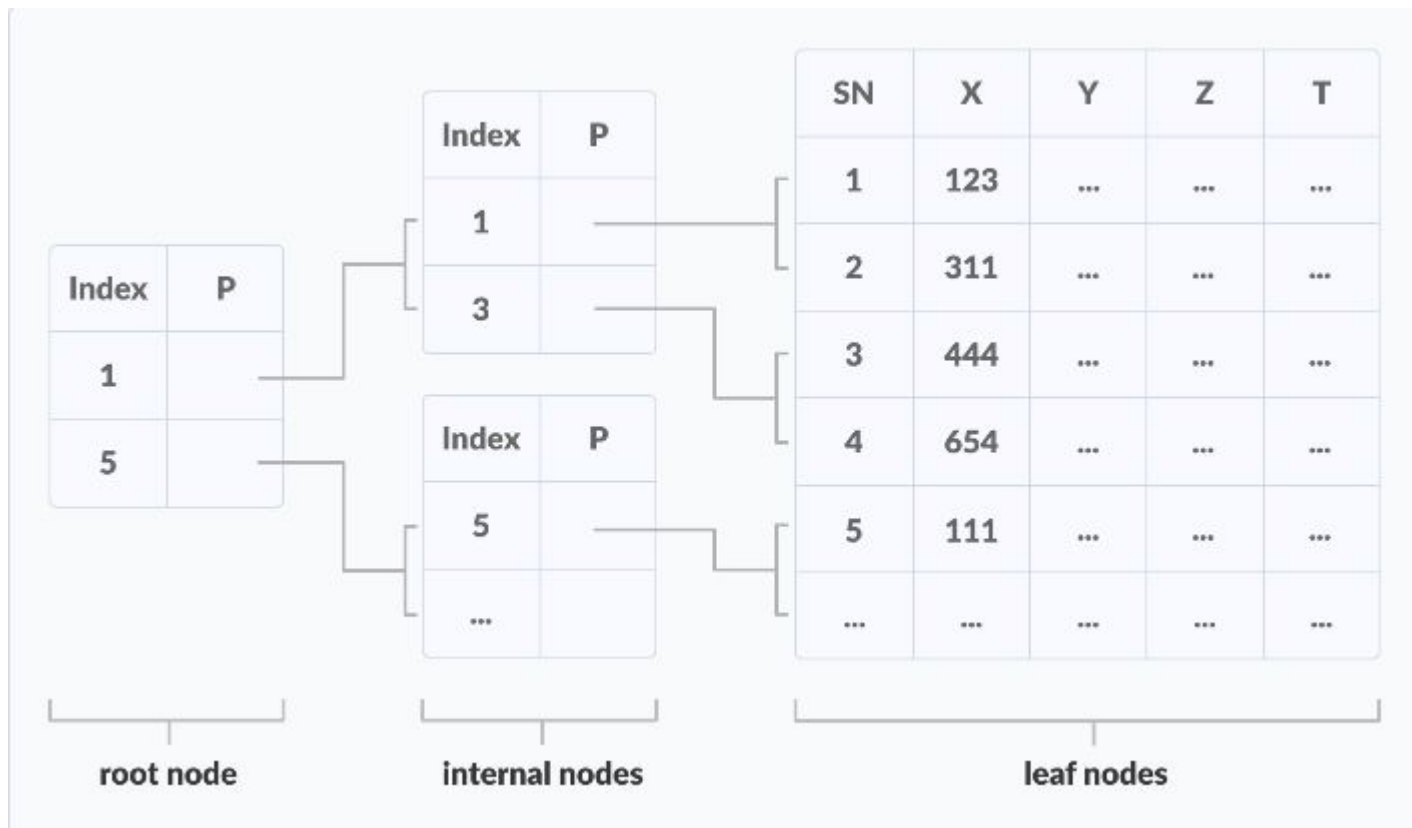


 Tree node pointer

 Data pointer

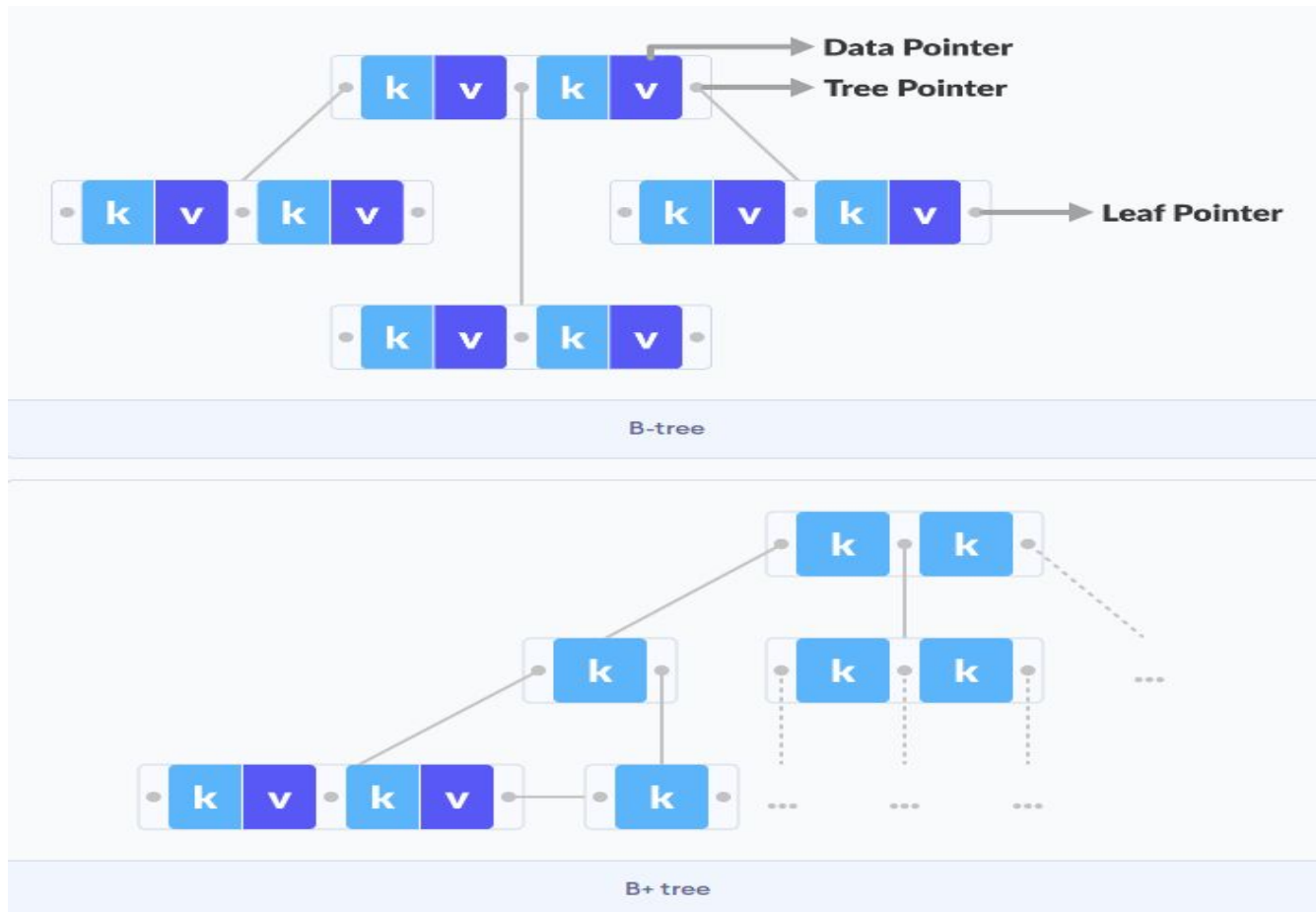
 Null pointer

B+ Tree



- https://www.cs.uct.ac.za/mit_notes/database/htmls/chp11.html#single-level-ordered-indexes

B+ Tree Vs. B Tree



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.

Why

- Key are primarily utilized to aid the search by directing to the proper Leaf.
- 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.

Properties

- Leaves are used to store data records.
- 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 $\lceil m/2 \rceil - 1$ keys.

Properties

- Leaves are used to store data records.
- 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.

Insert Operation

- The root has at least two children.
- Each node except root can have a maximum of m children and at least $\lceil m/2 \rceil$ children. ()
- Each node can contain a maximum of $m - 1$ keys and a minimum of $\lceil m/2 \rceil - 1$ keys.

Insert Operation

- 1,4,7,10,17,21,31,25,19,20,28,42
- Order is 4 ($m = 4$)
 - Max Children $\lceil \frac{m}{2} \rceil = 4$
 - Min Children $\lceil \frac{m}{2} \rceil = 2$
 - Max Key $\lceil \frac{m}{2} \rceil = 3$
 - Min Children $\lceil \frac{m}{2} \rceil = 1$

<https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html>

Delete Operation(1)

Order is 3 ($m = 3$)

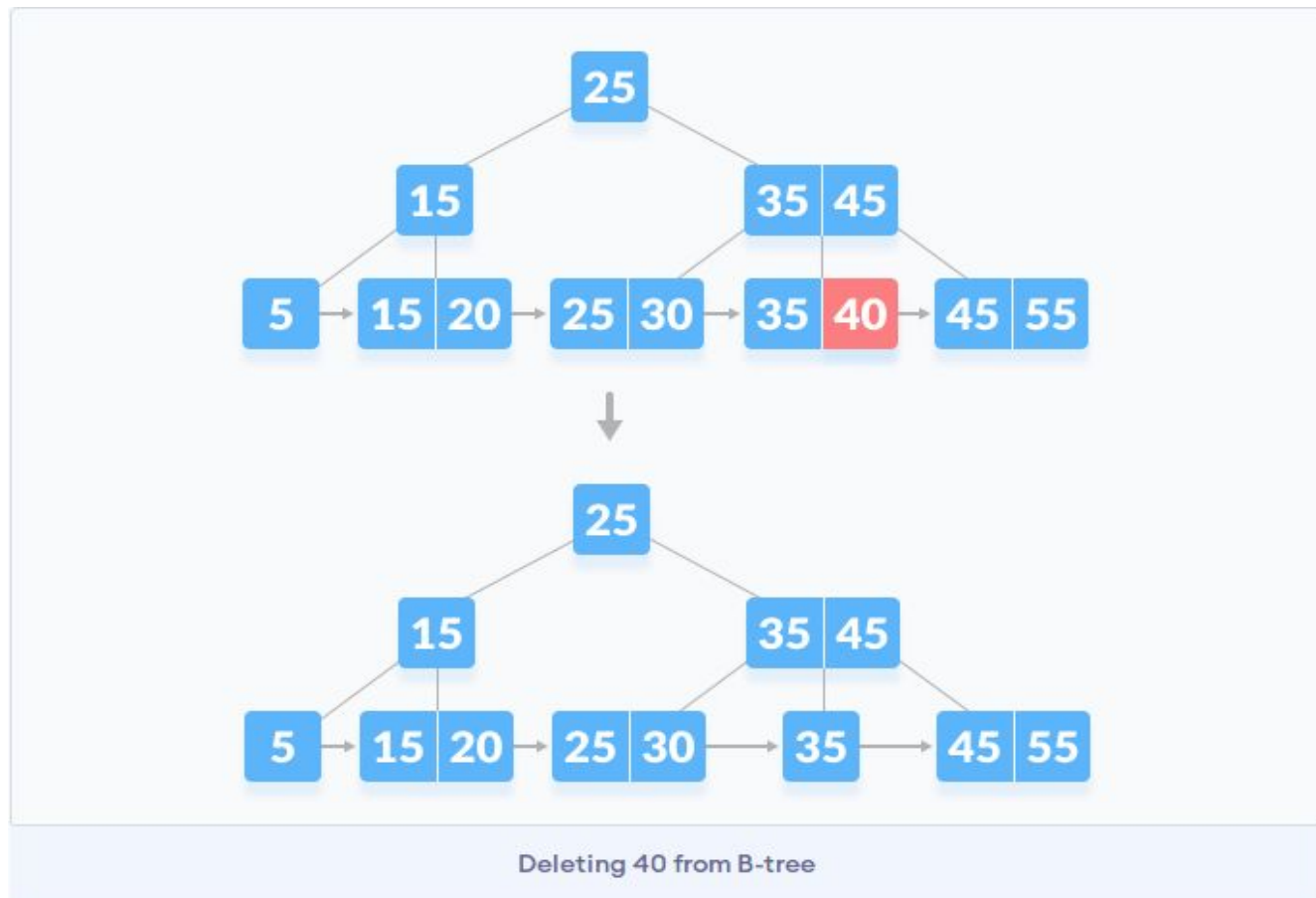
Max Children ≤ 3

Min Children ≥ 2

Max Key ≤ 2

Min Key ≥ 1

- Key \leq Only at the leaf node



Delete Operation(1)

Order is 3 ($m = 3$)

Max Children ≤ 3

Min Children ≥ 2

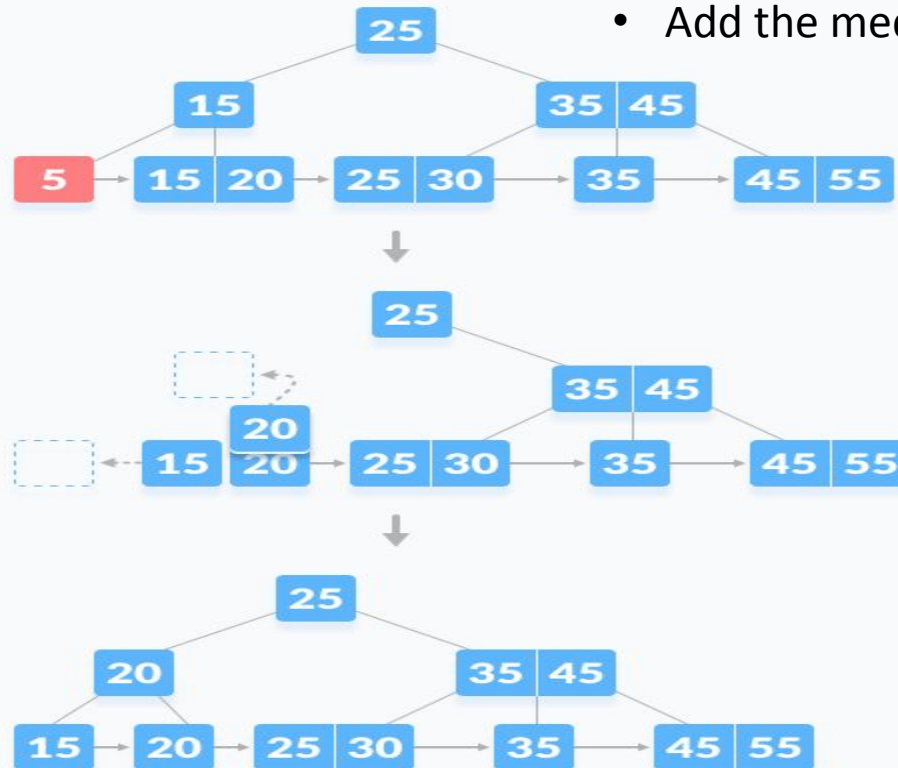
Max Key ≤ 2

Min Children ≥ 1

- Key \leq Only at the leaf node.

Delete the key and borrow a key from the sibling.

- Add the median key of the sibling node



Deleting 5 from B-tree

Order is 3 ($m = 3$)

Max Children ≥ 3

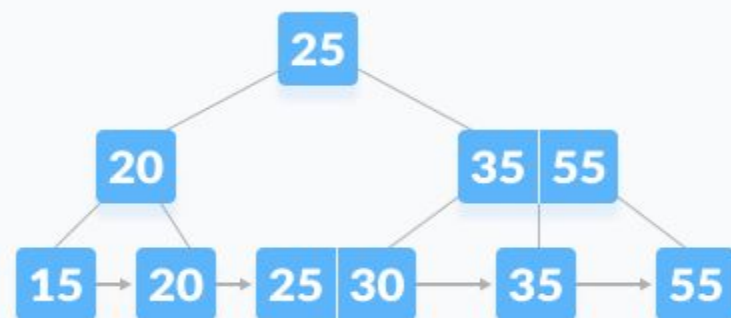
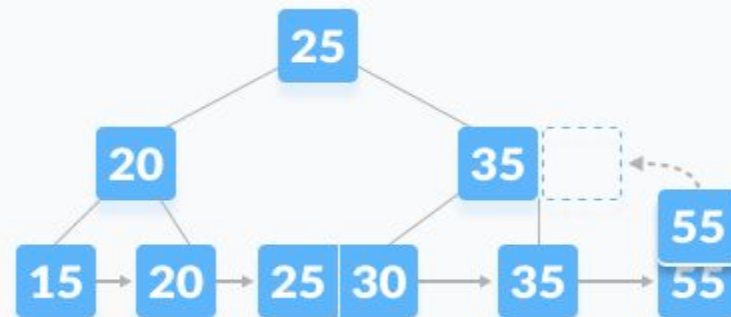
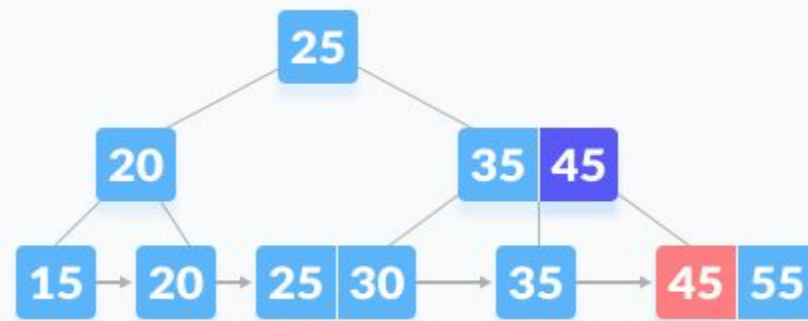
Min Children ≥ 2

Max Key ≥ 2

Min Children ≥ 1

Delete Operation (2)

- Key \geq leaf node + Internal Node
 - 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

Order is 3 ($m = 3$)

Max Children ≥ 3

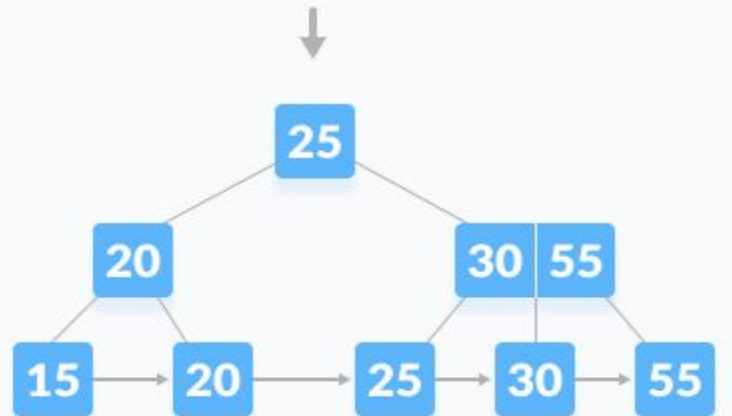
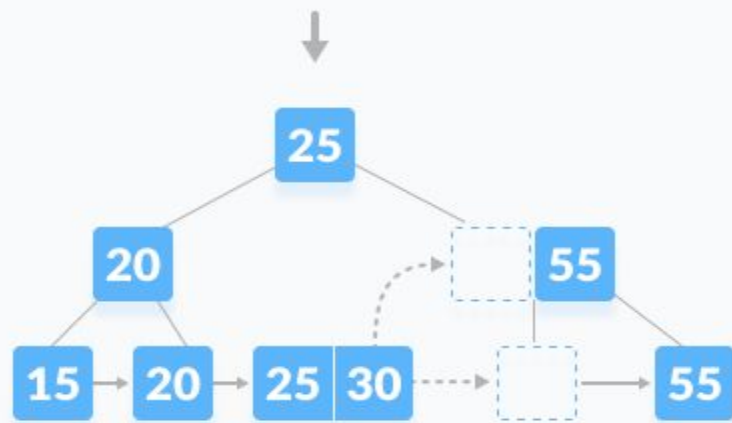
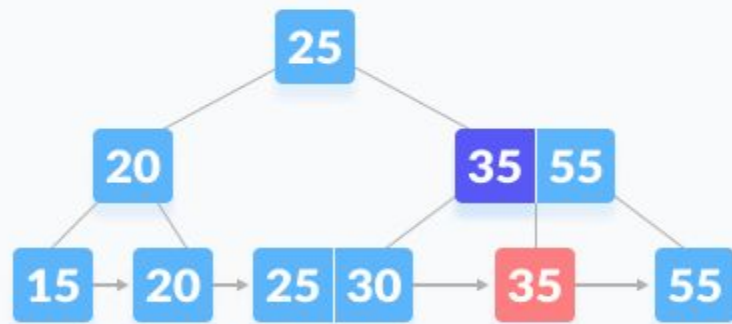
Min Children ≥ 2

Max Key ≥ 2

Min Children ≥ 1

Delete Operation (2)

- Key \geq leaf node + Internal Node
 - If there is an exact minimum number of keys in the node
 - Delete the key and borrow a key from its immediate sibling (through the
 - Fill the empty space created in the index (internal node) with the borrowed key. parent).

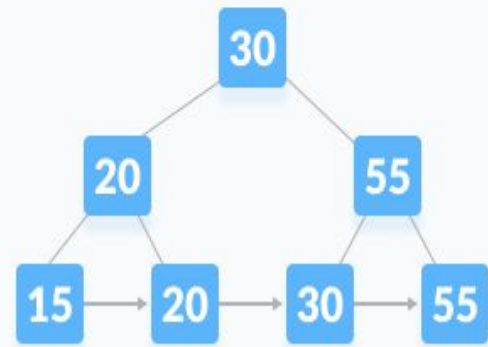
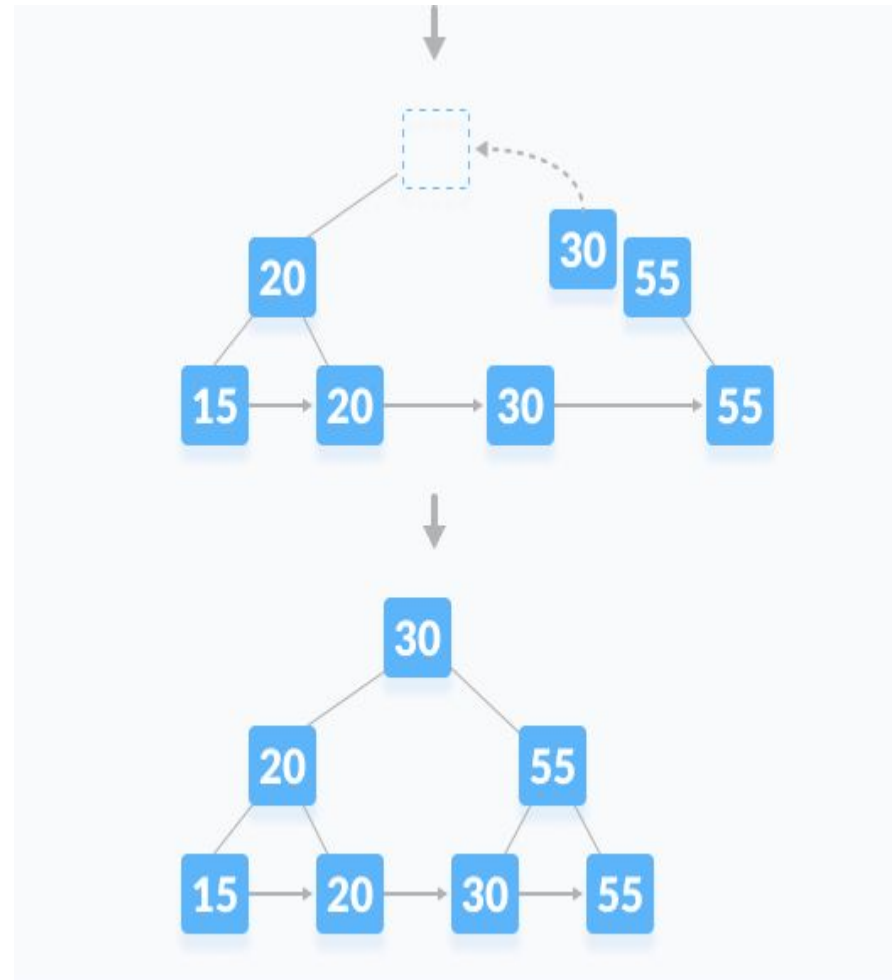
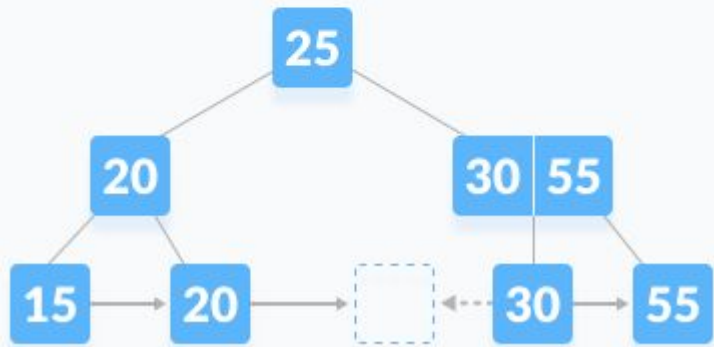
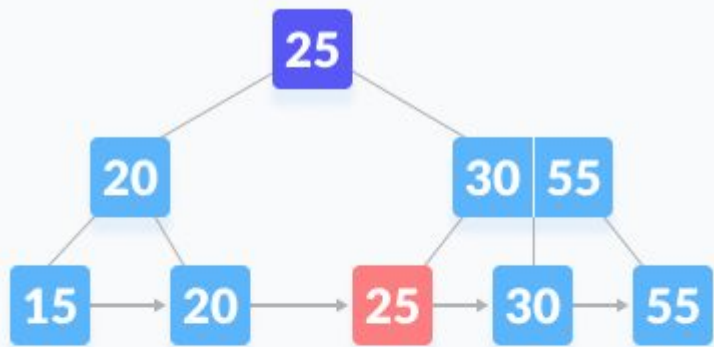


Deleting 35 from B-tree

Delete Operation (2)

Order is 3 ($m = 3$)
Max Children ≤ 3
Min Children ≥ 2
Max Key ≤ 2
Min Children ≥ 1

- Key \leq leaf node + Internal Node
 - 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.



Deleting 25 from B-tree

Delete Operation (3)

Order is 3 ($m = 3$)

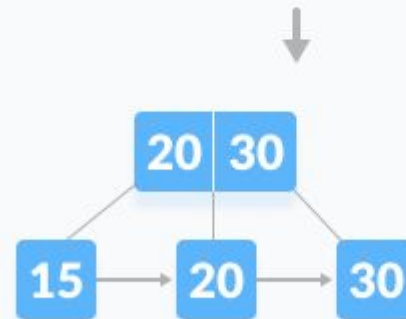
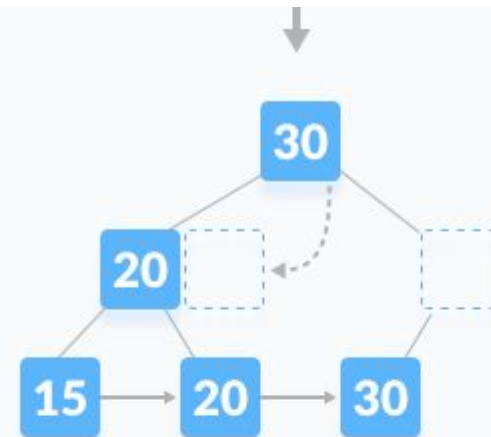
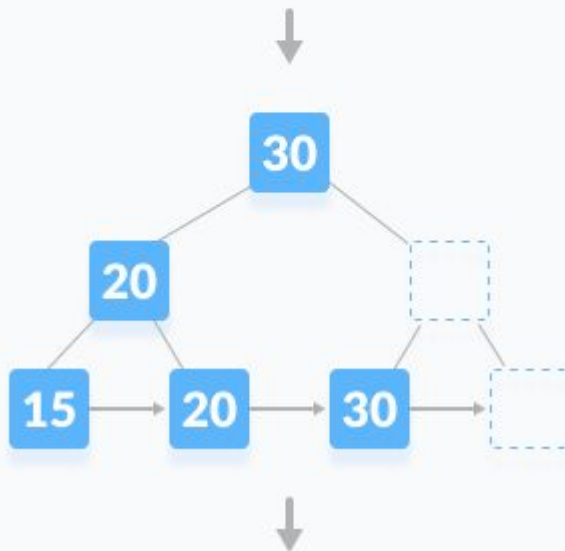
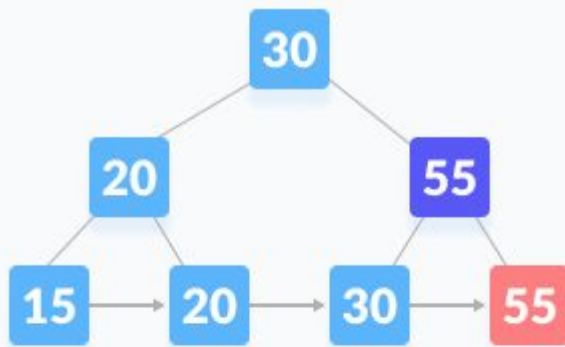
Max Children ≥ 3

Min Children ≥ 2

Max Key ≥ 2

Min Children ≥ 1

- Shrinking



Deleting 55 from B-tree

- F, S, Q, K, C, L, H, T, V, W, M, R, N, P, A, B, X, Y,
D, Z ,E