

# Insertion in B+ Tree and B- Tree

## B+ Tree Insertion

A **B+ Tree** is a self-balancing tree data structure that maintains sorted data and allows searches, sequential access, insertions, and deletions in logarithmic time.

### Steps to Insert a Node in a B+ Tree:

#### 1. Find the Correct Leaf Node:

- Start from the root and traverse down the tree.
- Use key comparisons to determine the correct child to follow.
- Continue until reaching the appropriate leaf node.

#### 2. Insert the Key in the Leaf Node:

- If the node has space, insert the key in the correct position (maintaining order) and update pointers.
- If the node is full, proceed to step 3.

#### 3. Handle Overflow:

- Split the leaf node into two nodes:
  - The left node keeps the lower half of the keys.
  - The right node takes the upper half of the keys.
- The middle key is promoted to the parent.
- If the parent is full, recursively split the parent.

#### 4. Adjust Internal Nodes and Root:

- If the root is split, a new root is created, increasing the tree height.
- Ensure all pointers in internal nodes correctly point to the new children.

## B- Tree Insertion

A **B- Tree** is a balanced search tree where all leaf nodes are at the same level, and each node has multiple children (determined by a minimum degree  $t$ ).

### Steps to Insert a Node in a B- Tree:

#### 1. Find the Correct Node:

- Traverse the tree from the root, choosing the appropriate child at each level based on key comparisons.
- 2. **Insert the Key in the Node:**
  - If the node has space, insert the key at the correct position.
  - If the node is full, proceed to step 3.
- 3. **Split the Node:**
  - Divide the full node into two nodes.
  - The middle key is moved up to the parent.
  - The left and right nodes retain the lower and upper half of the keys, respectively.
  - If the parent is full, recursively split the parent.
- 4. **Update the Root if Necessary:**
  - If the root node is split, a new root is created, increasing the height of the tree.

### Differences Between B+ Tree and B- Tree Insertions:

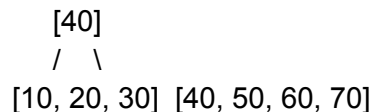
- In **B+ Trees**, all keys are stored in leaf nodes, and internal nodes only act as guides.
- In **B- Trees**, keys exist in both internal and leaf nodes.
- **B+ Trees** use linked leaf nodes for efficient range queries.

Both trees ensure logarithmic time complexity  $O(\log n)$  for insertions, maintaining balance dynamically to optimize search and retrieval operations.

## Examples of B+ Tree and B- Tree Insertions

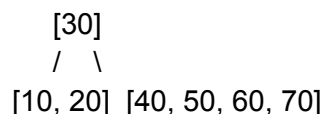
### Example 1:

Insert the values [30, 10, 20, 50, 40, 60, 70] into a B+ Tree with  $t=3$



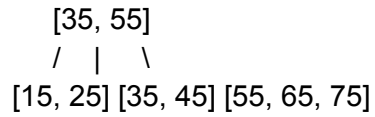
### Example 2:

Insert the values [30, 10, 20, 50, 40, 60, 70] into a B- Tree with  $t=3$



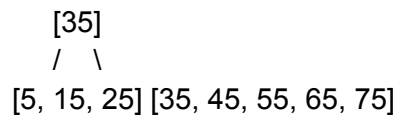
### Example 3:

Insert [15, 25, 35, 45, 55, 65, 75] into a B+ Tree with  $t=3$



### Example 4:

Insert [5, 15, 25, 35, 45, 55, 65, 75] into a B- Tree with  $t=3$



### Example 5:

Insert [10, 20, 30, 40, 50, 60, 70, 80, 90] into a B+ Tree with  $t=3$

