B+ Tree Walkthrough Mark Fontenot

Introduction to B+ Trees

A B+ tree is a balanced tree data structure commonly used in databases and file systems to efficiently store and retrieve data. In a B+ tree, all data values are stored in the leaf nodes, while internal nodes only contain keys and pointers for navigation.

Key Features:

- Nodes contain keys and pointers.
- Data is stored in leaf nodes.
- Internal nodes act solely as index structures.
- Efficiently supports insertions, deletions, and searches.

Example Walkthrough

B+ Tree (Order m = 4)

We start by inserting the keys:

Insert: 42, 21, 63, 89

- Initially, a single node acts as both the root and leaf node.
- Keys are inserted in sorted order: [21, 42, 63, 89].
- Each leaf node stores keys and associated data (data is omitted here for simplicity).
- Any further insertion into this node would require a split, as the node is at maximum capacity.

Insert: 35

- The leaf node is full; we must split it.
- A new leaf node is created to the right, and keys are redistributed:
  - Original node: [21, 35]
  - New node: [42, 63, 89]
- The smallest value from the new node (42) is copied up to create a new internal parent node.
- The structure now has an internal root node and two leaf nodes.

Insert: 10, 27, 96

- Begin at root (42). Each key directs us to a leaf node:
  - 10 < 42; insert into left leaf: [10, 21, 27, 35].
  - 96 > 42; insert into right leaf: [42, 63, 89, 96].
- No splits required, as both leaf nodes accommodate the insertions.

Insert: 30

- Navigate to the left leaf node, which is full ([10, 21, 27, 35]).
- Node splits, creating a new leaf node:
  - Original node: [10, 21]
  - New node: [27, 30, 35]
- Update leaf pointers to maintain a doubly linked list.

- Smallest key of the new node (27) is copied up to the internal parent node.
- Internal root node now contains [27, 42].

Advanced Insertion Example (Fast Forward)

The tree has grown, and the root node is at full capacity ([10, 27, 42, 63]). Additional insertions will deepen the tree structure.

Insert: 37 (Step 1)

- Navigate to correct leaf node, which is full ([27, 30, 35, 38]).
- Node splits:
  - Original node: [27, 30]
  - New node: [35, 37, 38]
- Smallest key (35) must be copied to the parent node.
- Parent node ([10, 27, 42, 63]) is also full, requiring further handling.

Insert: 37 (Step 2 – Splitting Internal Node)

- Internal node splits by moving middle key (35) upwards:
  - Left node: [10, 27]
  - Right node: [42, 63]
  - New root created with key [35].
- Tree height increases by one.

Final Structure

The tree is now deeper, balancing insertions efficiently across nodes. This walkthrough illustrates how a B+ tree maintains balance through splits and re-distributions, ensuring operations remain efficient even as data scales.