Chapter 12: Indexing – B-Trees and B+ Trees

12.6 B-Trees

Overview
B-trees are balanced tree data structures commonly attributed to R. Bayer and E. McCreight (1972). They significantly improved large-file storage and retrieval, largely replacing methods other than hashing by 1979. Their strength lies in efficiently supporting insertion, deletion, and key range searches.

Properties and Advantages of B-Trees
- **Shallow Structure:** Height-balanced with all leaves at the same level.
- **High Branching Factor:** Limits the number of disk accesses, enhancing performance.
- **Localized Updates:** Operations only affect nodes along the path from the root to a leaf.
- **Efficient Range Searches:** Related records are grouped together.
- **Space Efficiency:** Nodes must maintain a minimum occupancy, ensuring efficient space utilization.

Shape Properties of B-Trees
A B-tree of order `m` adheres to:
- Root is either a leaf or has at least two children.
- Internal nodes (except root) have between ⌈m/2⌉ and m children.
- Leaves are uniformly at the same level.

B-trees generalize 2-3 trees (B-trees of order three), and node sizes typically match disk blocks for optimized I/O operations.

Searching in B-Trees
- Begin at root; use binary search.
- Follow pointers according to comparison until a leaf is reached or key is found.

Insertion in B-Trees
- Locate the appropriate leaf node.
- Insert directly if space is available.
- If full, split the node and promote the middle key.
- Propagate splits upwards if necessary, maintaining balance.

12.6.1.1 B+ Trees

Motivation and Concept
B+ trees improve B-trees by storing records exclusively in leaf nodes, using internal nodes only for guidance. B+ trees are particularly efficient for range

queries due to linked leaf nodes.

Structure and Properties of B+ Trees
- Internal nodes contain keys and pointers.
- Leaf nodes store actual records or keys linked to external records.
- Leaf nodes form a doubly linked list.

Operations in B+ Trees
- **Search:** Always ends at leaf nodes. Internal nodes only direct searches.
- **Insertion:** Similar to B-tree insertion; leaf nodes are split evenly, and the smallest key from the right node is promoted.
- **Deletion:** Remove the record; if underflow occurs, redistribute from siblings or merge nodes.

Efficiency and Utilization
- B+ trees maintain at least 50% node occupancy.
- Enhanced variants like B* trees optimize storage further, keeping nodes about two-thirds full through more sophisticated splitting/merging strategies.

12.6.1.2 B-Tree Analysis

Efficiency
- Operations (search, insertion, deletion) in B-trees and variants (B+, B*) have complexity $\Theta(\log n)$, where n is the number of records.
- High branching factors make trees shallow, reducing disk accesses.

Practical Example
A B+ tree of order 100:
- Height 1: Up to 100 records.
- Height 2: Up to 10,000 records.
- Height 3: Up to 1,000,000 records.
- Height 4: Up to 100,000,000 records.

Disk Management
- Top levels can be kept in main memory to minimize disk fetches.
- A buffer pool (using an LRU policy or similar) manages nodes in memory, further optimizing performance.

Conclusion
B-trees and B+ trees are robust data structures, essential for database management, providing balanced and efficient operations tailored for disk storage systems.