B + Tree is a variation of the B-tree data structure. In a B + tree, data pointers are stored only at the leaf nodes of the tree. In a B+ tree structure of a leaf node differs from the structure of internal nodes. The leaf nodes have an entry for every value of the search field, along with a data pointer to the record (or to the block that contains this record). The leaf nodes of the B+ tree are linked together to provide ordered access to the search field to the records. Internal nodes of a B+ tree are used to guide the search. Some search field values from the leaf nodes are repeated in the internal nodes of the B+ tree.

Features of B+ Trees

- Balanced: B+ Trees are self-balancing, which means that as data is added or removed from the tree, it automatically adjusts itself to maintain a balanced structure. This ensures that the search time remains relatively constant, regardless of the size of the tree.
- Multi-level: B+ Trees are multi-level data structures, with a root node at the top and one or more levels of internal nodes below it. The leaf nodes at the bottom level contain the actual data.
- Ordered: B+ Trees maintain the order of the keys in the tree, which makes it easy to perform range queries and other operations that require sorted data.
- Fan-out: B+ Trees have a high fan-out, which means that each node can have many child nodes. This reduces the height of the tree and increases the efficiency of searching and indexing operations.
- Cache-friendly: B+ Trees are designed to be cache-friendly, which means that they can take advantage of the caching mechanisms in modern computer architectures to improve performance.
- Disk-oriented: B+ Trees are often used for disk-based storage systems because they are efficient at storing and retrieving data from disk.

Why Use B+ Tree?

- B+ Trees are the best choice for storage systems with sluggish data access because they minimize I/O operations while facilitating efficient disc access.
- B+ Trees are a good choice for database systems and applications needing quick data retrieval because of their balanced structure, which guarantees predictable performance for a variety of activities and facilitates effective range-based queries.

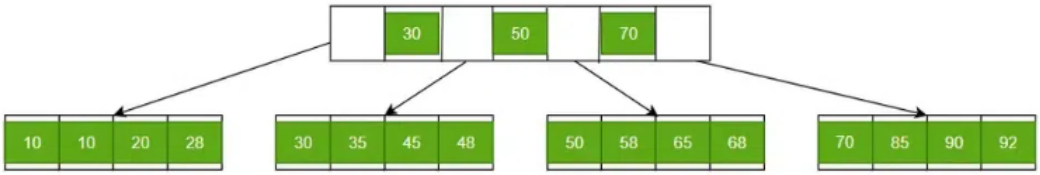
Difference Between B+ Tree and B Tree

Parameters	B+ Tree	B Tree
Structure	Separate leaf nodes for data storage and internal nodes for indexing	Nodes store both keys and data values

Leaf Nodes	Leaf nodes form a linked list for efficient range-based queries	Leaf nodes do not form a linked list
Order	Higher order (more keys)	Lower order (fewer keys)
Key Duplication	Typically allows key duplication in leaf nodes	Usually does not allow key duplication
Disk Access	Better disk access due to sequential reads in a linked list structure	More disk I/O due to non-sequential reads in internal nodes
Applications	Database systems, file systems, where range queries are common	In-memory data structures, databases, general-purpose use
Performance	Better performance for range queries and bulk data retrieval	Balanced performance for search, insert, and delete operations
Memory Usage	Requires more memory for internal nodes	Requires less memory as keys and values are stored in the same node

Some differences between B+ Tree and B Tree are stated below.Implementation of B+ Tree
In order, to implement dynamic multilevel indexing, B-tree and B+ tree are generally employed. The drawback of the B-tree used for indexing, however, is that it stores the data pointer (a pointer to the disk file block containing the key value), corresponding to a particular key value, along with that key value in the node of a B-tree. This technique greatly reduces the number of entries that can be packed into a node of a B-tree, thereby contributing to the increase in the number of levels in the B-tree, hence increasing the search time of a record. B+ tree eliminates the above drawback by storing data pointers only at the leaf nodes of the tree. Thus, the structure of the leaf nodes of a B+ tree is quite different from the structure of the internal nodes of the B tree. It may be noted here that, since data pointers are present only at the leaf nodes,

the leaf nodes must necessarily store all the key values along with their corresponding data pointers to the disk file block, in order to access them.
Moreover, the leaf nodes are linked to providing ordered access to the records. The leaf nodes, therefore form the first level of the index, with the internal nodes forming the other levels of a multilevel index. Some of the key values of the leaf nodes also appear in the internal nodes, to simply act as a medium to control the searching of a record. From the above discussion, it is apparent that a B+ tree, unlike a B-tree, has two orders, ‘a’ and ‘b’, one for the internal nodes and the other for the external (or leaf) nodes.
Structure of B+ Trees



B+ Trees contain two types of nodes:

- Internal Nodes: Internal Nodes are the nodes that are present in at least n/2 record pointers, but not in the root node,
- Leaf Nodes: Leaf Nodes are the nodes that have n pointers.

The Structure of the Internal Nodes of a B+ Tree of Order ‘a’ is as Follows

- Each internal node is of the form: <P1, K1, P2, K2, ….., Pc-1, Kc-1, Pc> where c <= a and each Pi is a tree pointer (i.e points to another node of the tree) and, each Ki is a key-value (see diagram-I for reference).
- Every internal node has : K1 < K2 < …. < Kc-1
- For each search field value ‘X’ in the sub-tree pointed at by Pi, the following condition holds: Ki-1 < X <= Ki, for 1 < I < c and, Ki-1 < X, for i = c (See diagram I for reference)
- Each internal node has at most ‘aa tree pointers.
- The root node has, at least two tree pointers, while the other internal nodes have at least \ceil(a/2) tree pointers each.
- If an internal node has ‘c’ pointers, c <= a, then it has ‘c – 1’ key values.

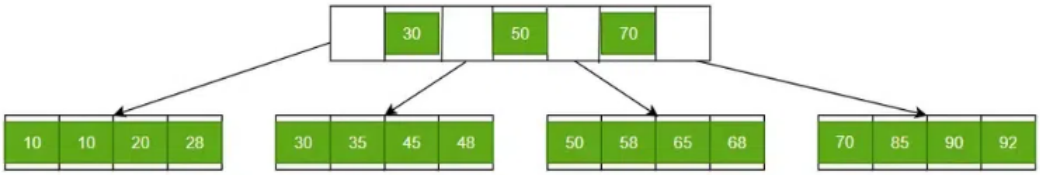
Structure of Internal Node

The Structure of the Leaf Nodes of a B+ Tree of Order ‘b’ is as Follows

- Each leaf node is of the form: <<K1, D1>, <K2, D2>, ….., <Kc-1, Dc-1>, Pnext> where c <= b and each Di is a data pointer (i.e points to actual record in the disk whose key value is Ki or to a disk file block containing that record) and, each Ki is a key value and, Pnext points to next leaf node in the B+ tree (see diagram II for reference).
- Every leaf node has : K1 < K2 < …. < Kc-1, c <= b
- Each leaf node has at least \ceil(b/2) values.
- All leaf nodes are at the same level.

Structure of Lead Node
Diagram-II Using the Pnext pointer it is viable to traverse all the leaf nodes, just like a linked list, thereby achieving ordered access to the records stored in the disk.

Tree Pointer
Searching a Record in B+ Trees



Searching in B+ Tree
Let us suppose we have to find 58 in the B+ Tree. We will start by fetching from the root node then we will move to the leaf node, which might contain a record of 58. In the image given above, we will get 58 between 50 and 70. Therefore, we will we are getting a leaf node in the third leaf node and get 58 there. If we are unable to find that node, we will return that 'record not founded' message.
Insertion in B+ Trees
Insertion in B+ Trees is done via the following steps.

- Every element in the tree has to be inserted into a leaf node. Therefore, it is necessary to go to a proper leaf node.
- Insert the key into the leaf node in increasing order if there is no overflow.

For more, refer to Insertion in a B+ Trees.
Deletion in B+Trees
Deletion in B+ Trees is just not deletion but it is a combined process of Searching, Deletion, and Balancing. In the last step of the Deletion Process, it is mandatory to balance the B+ Trees, otherwise, it fails in the property of B+ Trees.
For more, refer to Deletion in B+ Trees.
Advantages of B+Trees

- A B+ tree with 'l' levels can store more entries in its internal nodes compared to a B-tree having the same 'l' levels. This accentuates the significant improvement made to the search time for any given key. Having lesser levels and the presence of Pnext pointers imply that the B+ trees is very quick and efficient in accessing records from disks.
- Data stored in a B+ tree can be accessed both sequentially and directly.
- It takes an equal number of disk accesses to fetch records.
- B+trees have redundant search keys, and storing search keys repeatedly is not possible.

Disadvantages of B+ Trees

- The major drawback of B-tree is the difficulty of traversing the keys sequentially. The B+ tree retains the rapid random access property of the B-tree while also allowing rapid sequential access.