B-Trees And B+-Trees

Preview

- B-Tree Indexing
- B-Tree
- B-Tree Characteristics
- B-Tree Example
- B+-Tree
- B+-Tree Characteristics
- B+-Tree Example

B-Tree Index

- Standard use index in relational databases in a B-Tree index.
- Allows for rapid tree traversal searching through an upside-down tree structure
- Reading a single record from a very large table using a B-Tree index, can often result in a few block reads—even when the index and table are millions of blocks in size.
- Any index structure other than a B-Tree index is subject to overflow.
  - Overflow is where any changes made to tables will not have records added into the original index structure, but rather tacked on the end.

What is a B-Tree?

- B-tree is a specialized multiway tree designed especially for use on disk.
- B-Tree consists of a root node, branch nodes and leaf nodes containing the indexed field values in the ending (or leaf) nodes of the tree.

B-Tree Characteristics

- In a B-tree each node may contain a large number of keys
- B-tree is designed to branch out in a large number of directions and to contain a lot of keys in each node so that the height of the tree is relatively small
- Constraints that tree is always balanced
- Space wasted by deletion, if any, never becomes excessive
- Insert and deletions are simple processes
  - Complicated only under special circumstances

    -Insertion into a node that is already full or a deletion from a node makes it less then half full

Characteristics of a B-Tree of Order P

- Within each node, $K_1 < K_2 < .. < K_{p-1}$
- Each node has at most p tree pointer
- Each node, except the root and leaf nodes, has at least ceil(p/2) tree pointers, The root node has at least two tree pointers unless it is the only node in the tree.
- All leaf nodes are at the same level. Leaf node have the same structure as internal nodes except that all of their tree pointer P_i are null.

B-Tree Insertion

1) B-tree starts with a single root node (which is also a leaf node) at level 0.
2) Once the root node is full with p – 1 search key values and when attempt to insert another entry in the tree, the root node splits into two nodes at level 1.
3) Only the middle value is kept in the root node, and the rest of the values are split evenly between the other two nodes.
4) When a nonroot node is full and a new entry is inserted into it, that node is split into two nodes at the same level, and the middle entry is moved to the parent node along with two pointers to the new split nodes.
5) If the parent node is full, it is also split.
6) Splitting can propagate all the way to the root node, creating a new level if the root is split.

B-Tree Deletion

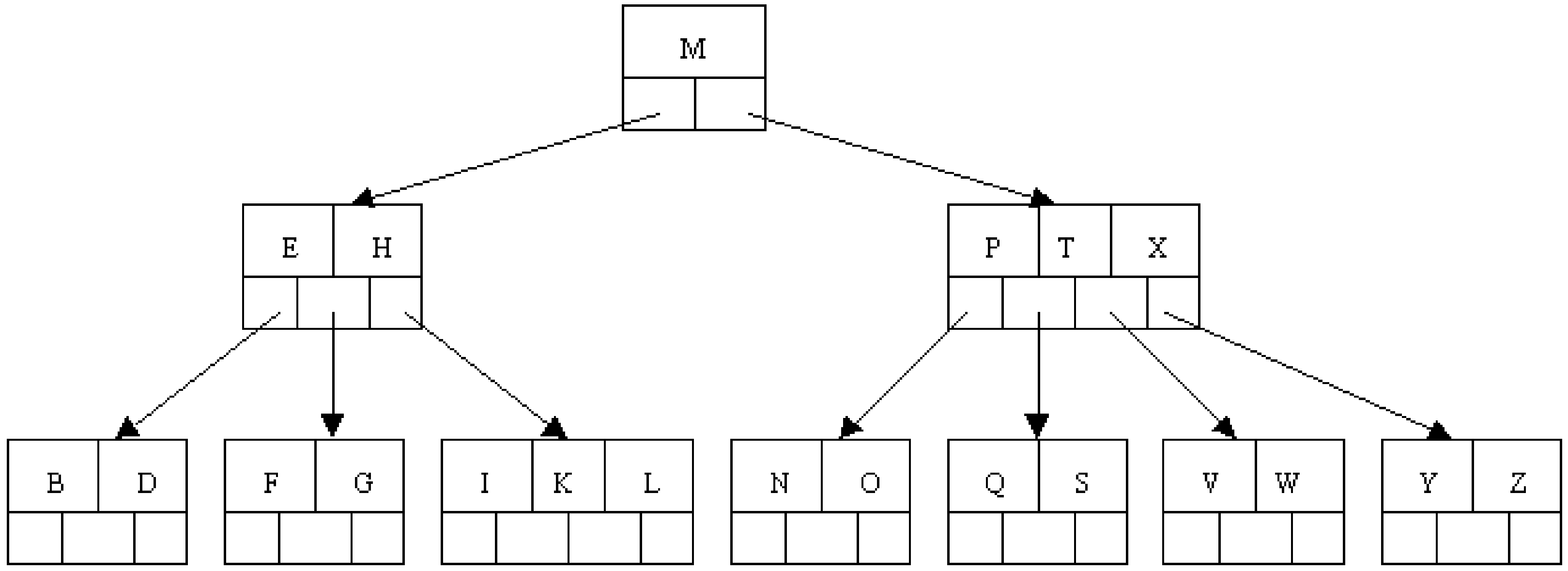
1) If deletion of a value causes a node to be less than half full, it is combined with it neighboring nodes, and this can also propagate all the way to the root.

   - Can reduce the number of tree levels.

*Shown by analysis and simulation that, after numerous random insertions and deletions on a B-tree, the nodes are approximately 69 percent full when the number of values in the tree stabilizes. If this happens , node splitting and combining will occur only rarely, so insertion and deletion become quite efficient.

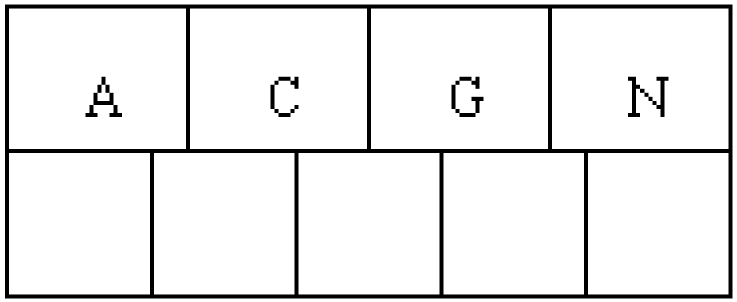
B-tree of Order 5 Example

- All internal nodes have at least ceil(5 / 2) = ceil(2.5) = 3 children (and hence at least 2 keys), other then the root node.
- The maximum number of children that a node can have is 5 (so that 4 is the maximum number of keys)
- each leaf node must contain at least 2 keys



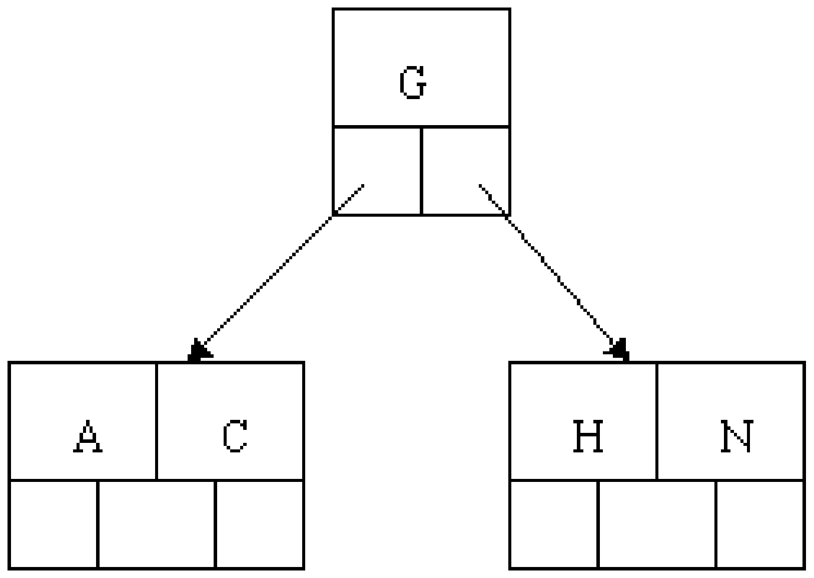
B-Tree Order 5 Insertion

- Originally we have an empty B-tree of order 5
- Want to insert C N G A H E K Q M F W L T Z D P R X Y S
- Order 5 means that a node can have a maximum of 5 children and 4 keys
- All nodes other than the root must have a minimum of 2 keys
- The first 4 letters get inserted into the same node



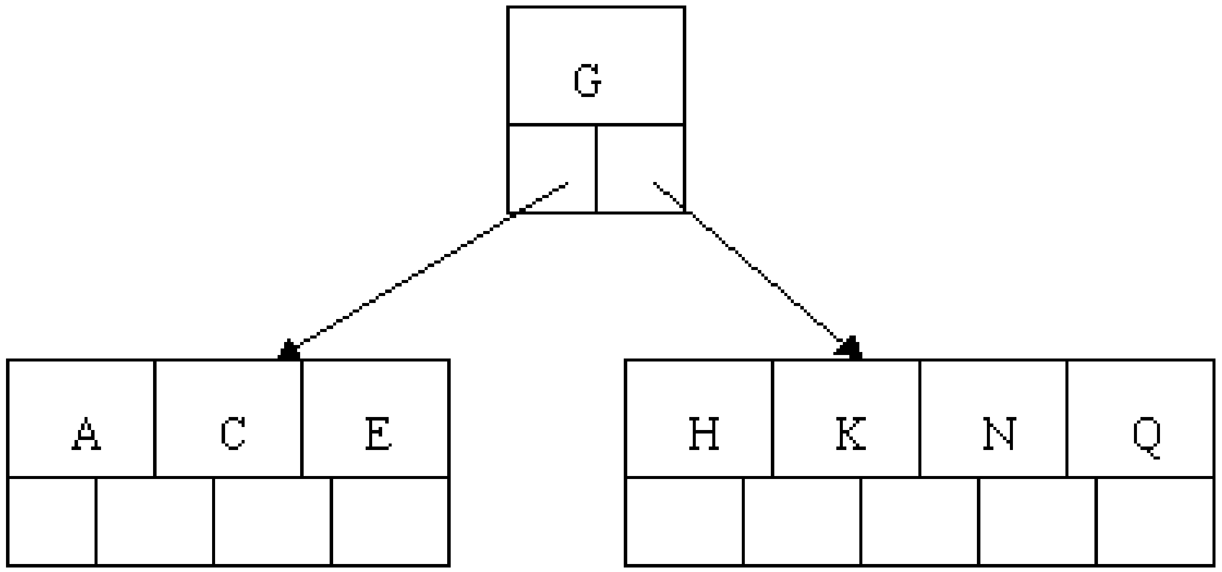
B-Tree Order 5 Insertion Cont.

- When we try to insert the H, we find no room in this node, so we split it into 2 nodes, moving the median item G up into a new root node.



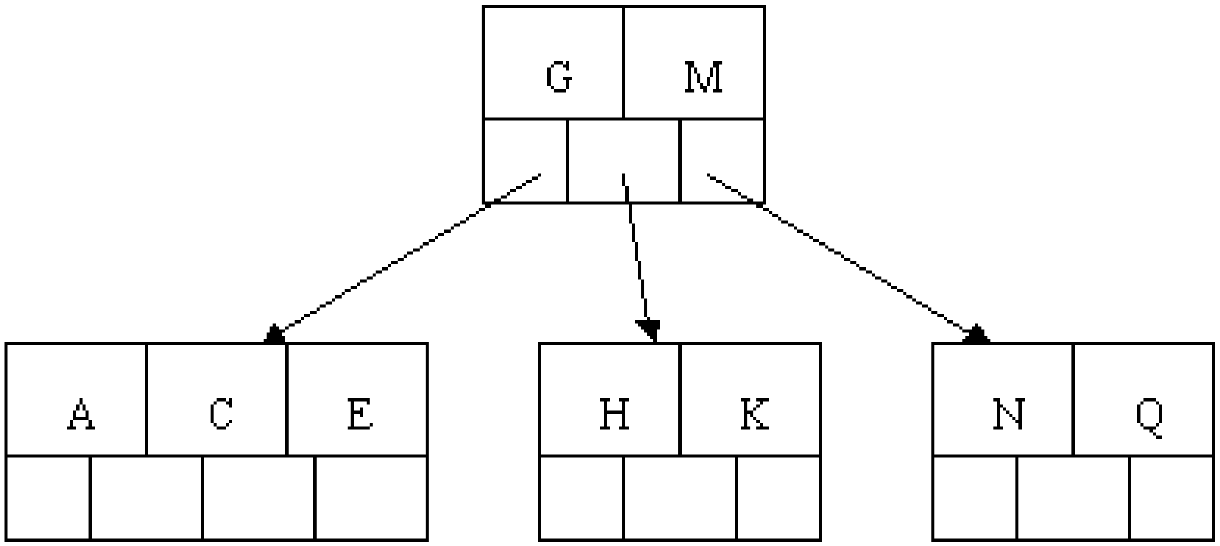
B-Tree Order 5 Insertion Cont.

- Inserting E, K, and Q proceeds without requiring any splits



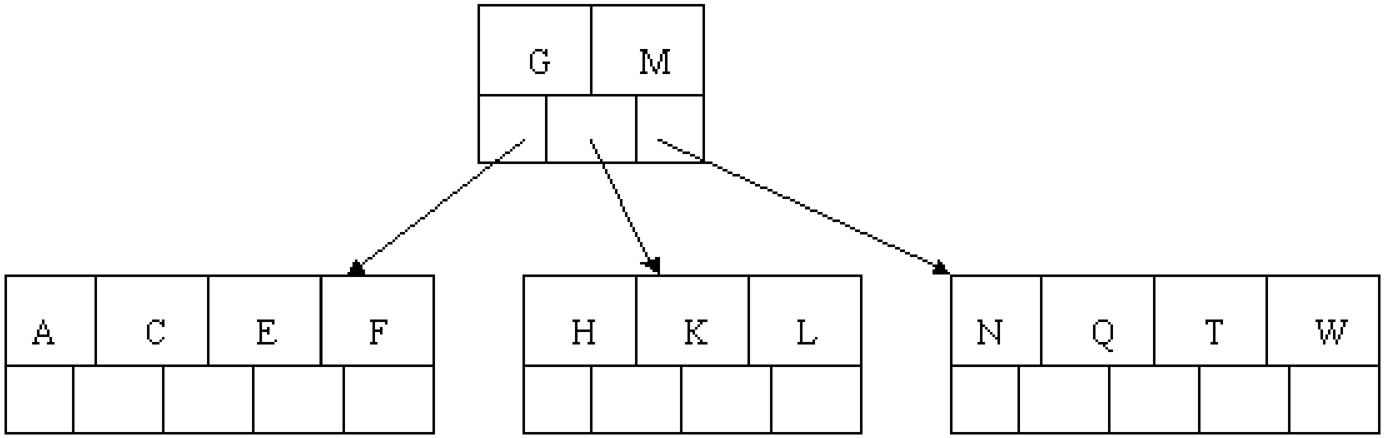
B-Tree Order 5 Insertion Cont.

- Inserting M requires a split



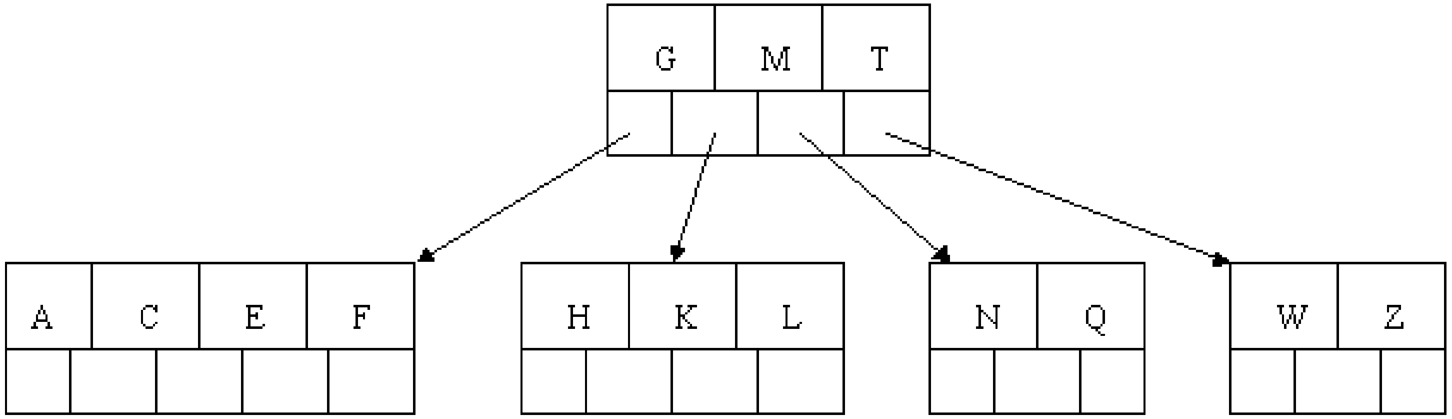
B-Tree Order 5 Insertion Cont.

- The letters F, W, L, and T are then added without needing any split



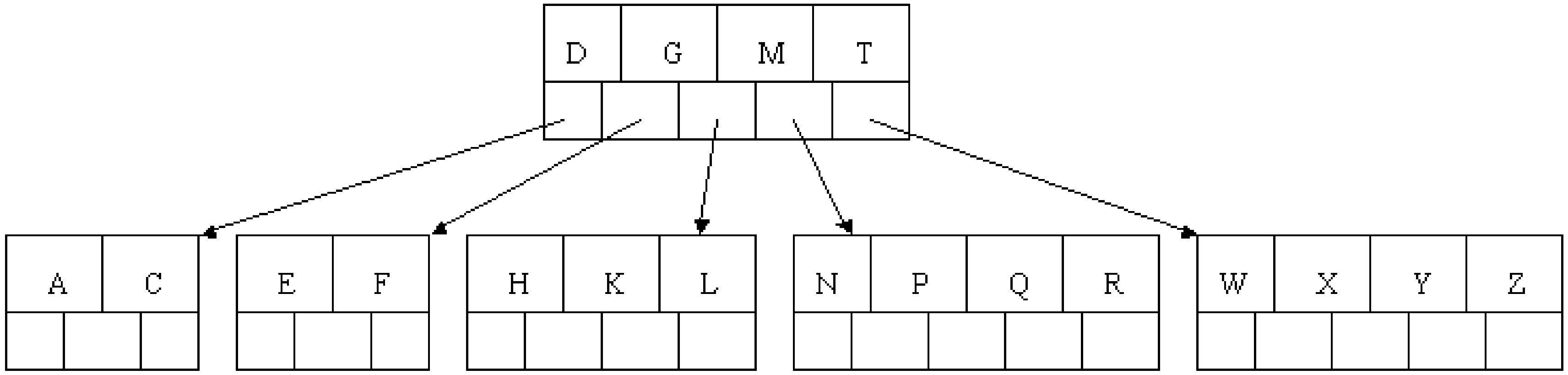
B-Tree Order 5 Insertion Cont.

- When Z is added, the rightmost leaf must be split. The median item T is moved up into the parent node



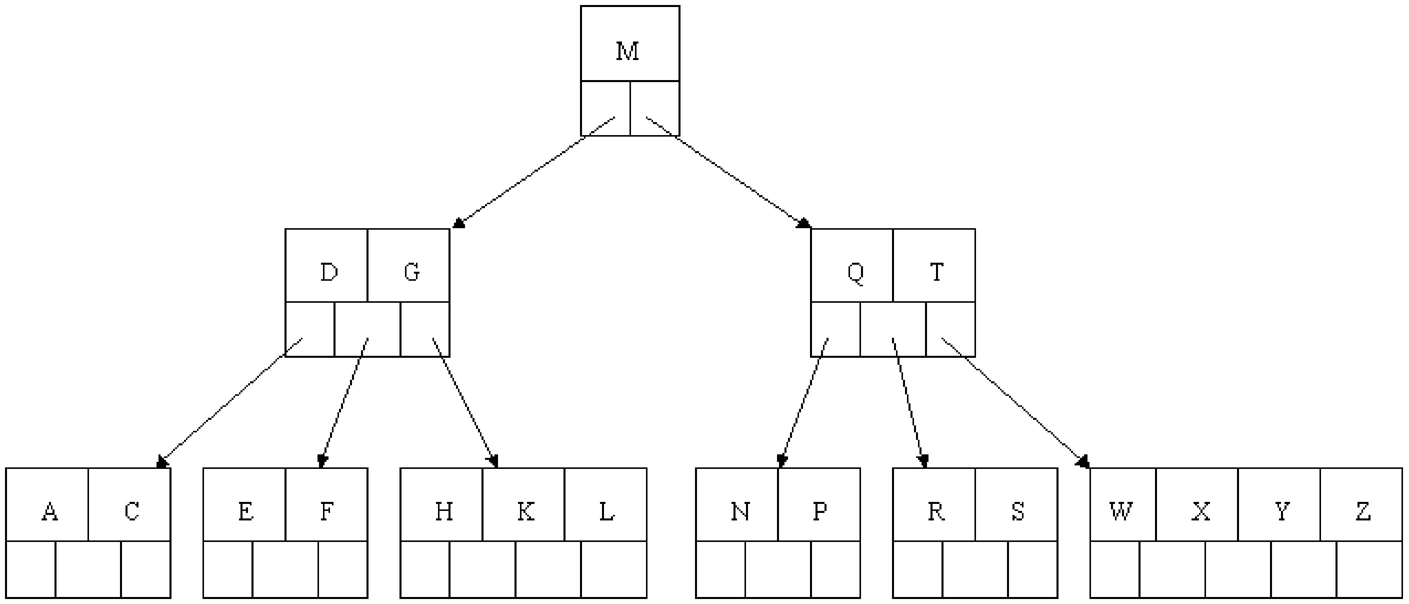
B-Tree Order 5 Insertion Cont.

- The insertion of D causes the leftmost leaf to be split. D happens to be the median key and so is the one moved up into the parent node.
- The letters P, R, X, and Y are then added without any need of splitting



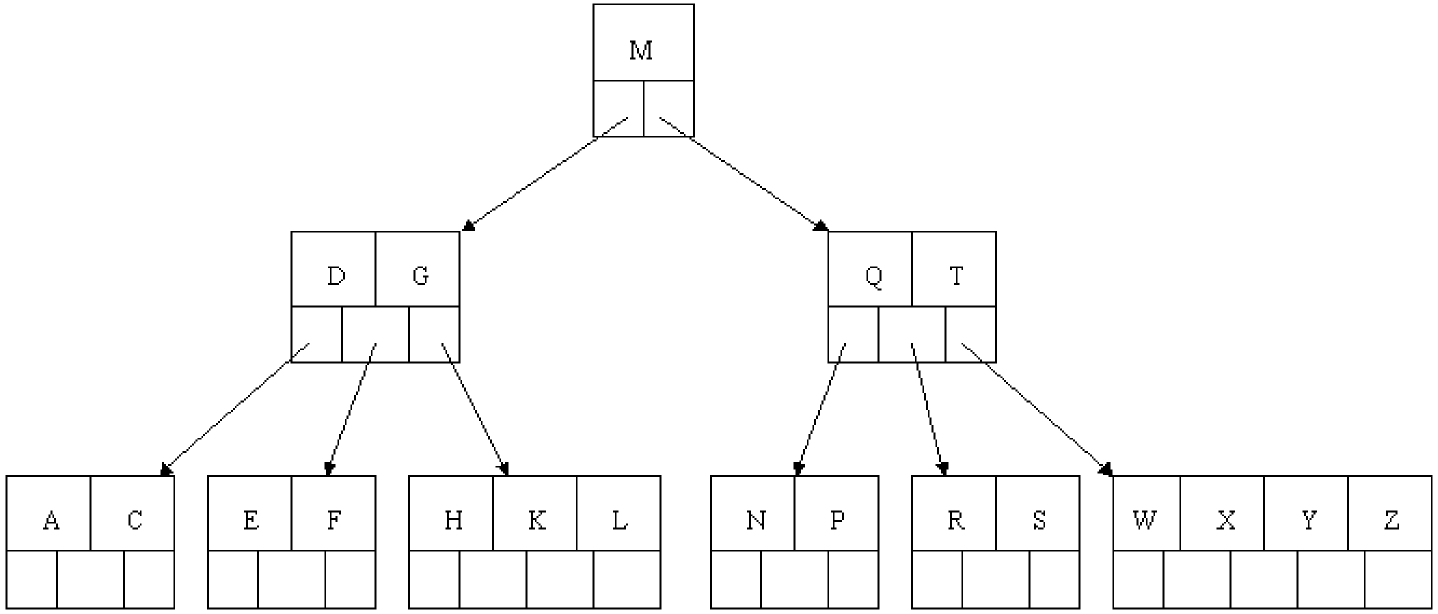
B-Tree Order 5 Insertion Cont.

- Finally, when S is added, the node with N, P, Q, and R splits, sending the median Q up to the parent.
- The parent node is full, so it splits, sending the median M up to form a new root node.



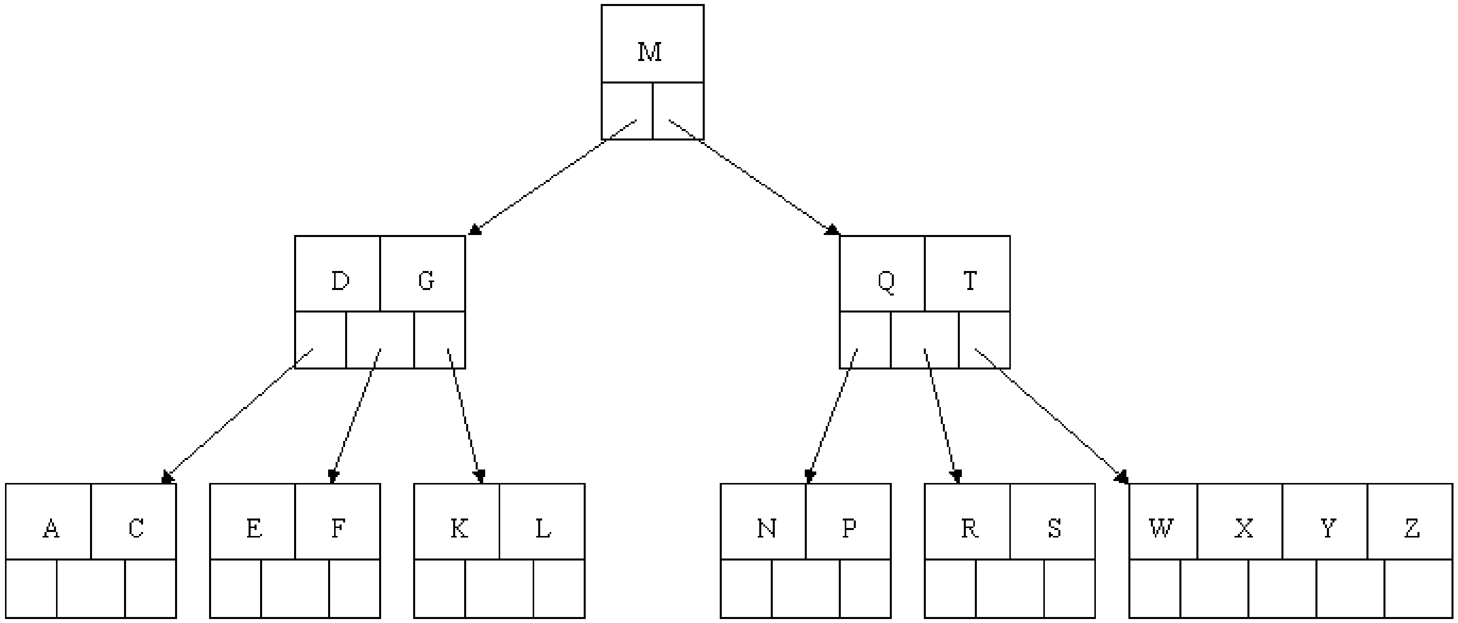
B-Tree Order 5 Deletion

- Initial B-Tree



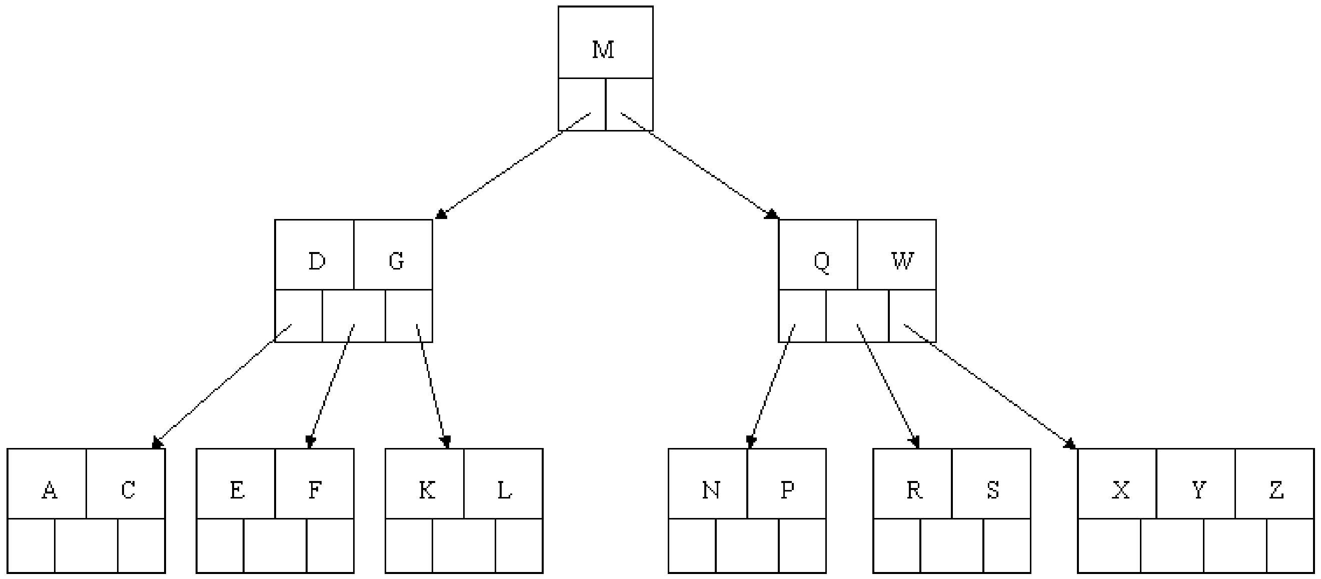
B-Tree Order 5 Deletion Cont.

- Delete H
- Since H is in a leaf and the leaf has more than the minimum number of keys, we just remove it.



B-Tree Order 5 Deletion Cont.

- Delete T.
- Since T is not in a leaf, we find its successor (the next item in ascending order), which happens to be W.
- Move W up to replace the T. That way, what we really have to do is to delete W from the leaf .



B+- Tree Characteristics

- Data records are only stored in the leaves.
- Internal nodes store just keys.
- Keys are used for directing a search to the proper leaf.
- If a target key is less than a key in an internal node, then the pointer just to its left is followed.
- If a target key is greater or equal to the key in the internal node, then the pointer to its right is followed.
- B+ Tree combines features of ISAM (Indexed Sequential Access Method) and B Trees.

B+- Tree Characteristics Cont.

- Implemented on disk, it is likely that the leaves contain key, pointer pairs where the pointer field points to the record of data associated with the key.
  - allows the data file to exist separately from the B+ tree, which functions as an "index" giving an ordering to the data in the data file.

B+- Tree Characteristics Cont.

- Very Fast Searching
- Insertion and deletion are expensive.

$$\left\lceil \log_{\left\lceil \frac{p}{2} \right\rceil} N \right\rceil$$

N	number of search values
p	order, number of block pointers per node

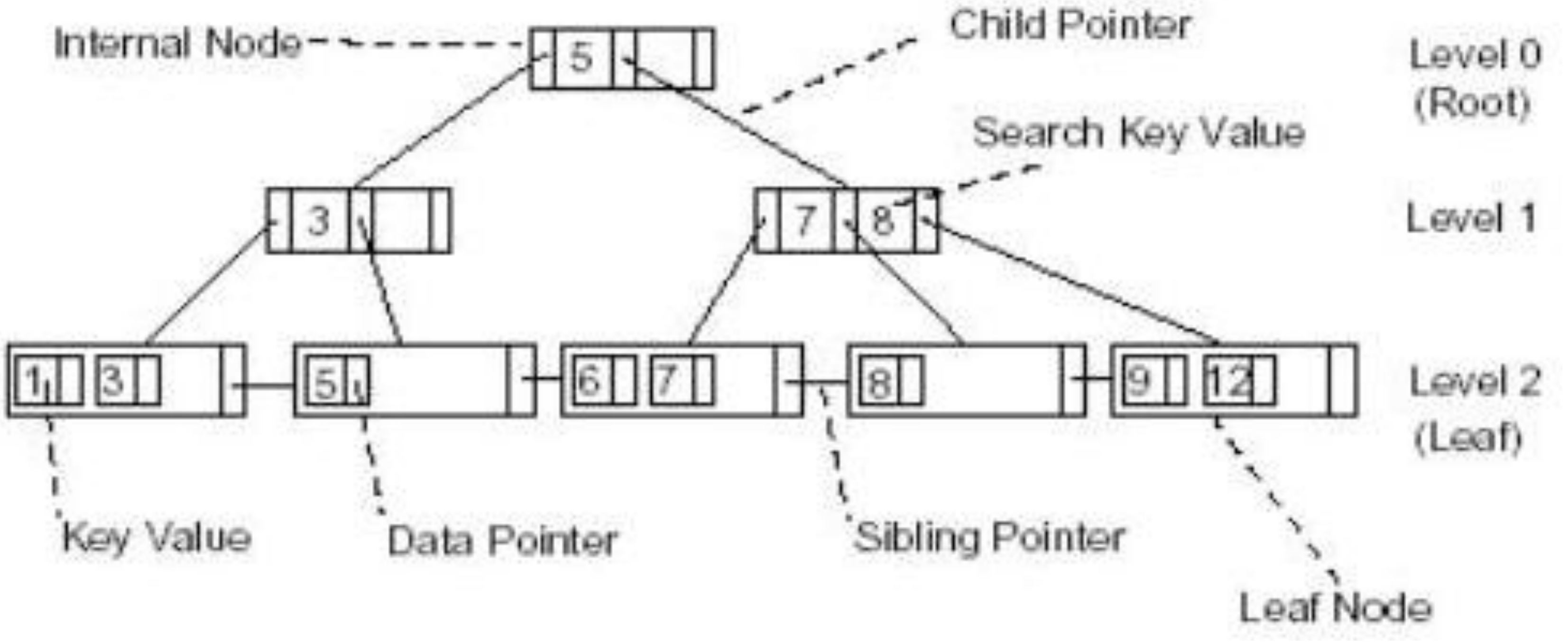
Formula n-order B+ tree with a height of h

- Maximum number of keys is n^h
- Minimum number of keys is $2(n / 2)^{h-1}$

B+ tree of order 200 Example

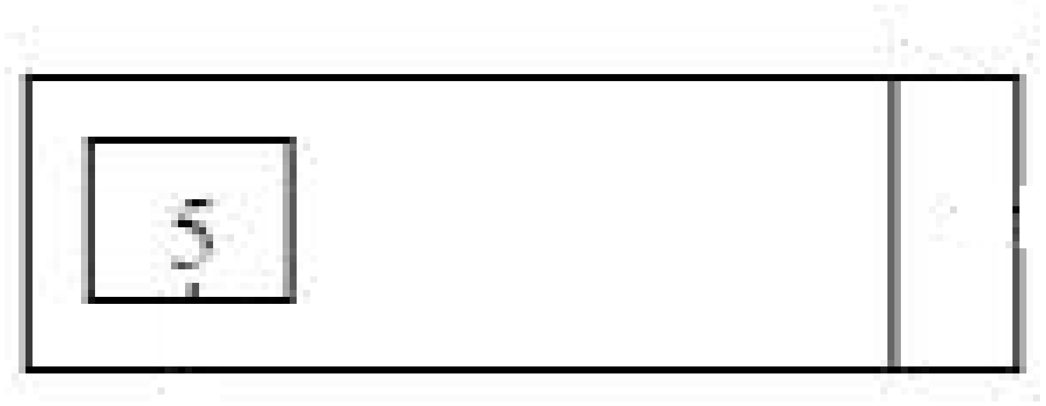
- Leaves can each contain up to 199 keys
- Assuming that the root node has at least 100 children
- A 2 level B+ tree that meets these assumptions can store about 9,900 records, since there are at least 100 leaves, each containing at least 99 keys.
- A 3 level B+ tree of this type can store about 1 million keys. A 4 level B+ tree can store up to about 100 million keys.

B+- Tree Structure



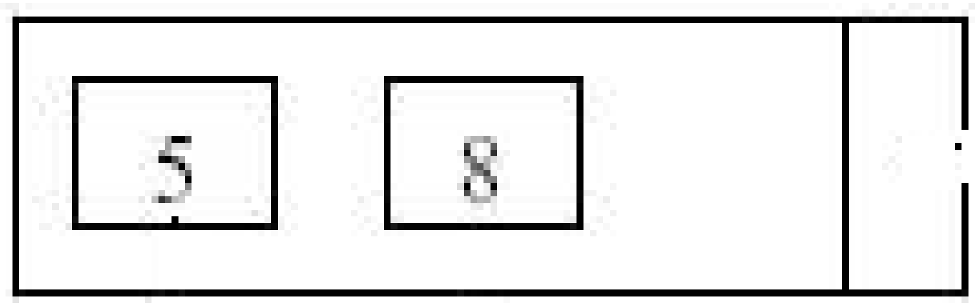
B+- Tree order 3 Insertion

- Insert value 5, 8, 1, 7
- Inserting value 5
- Since the node is empty, the value must be placed in the leaf node.



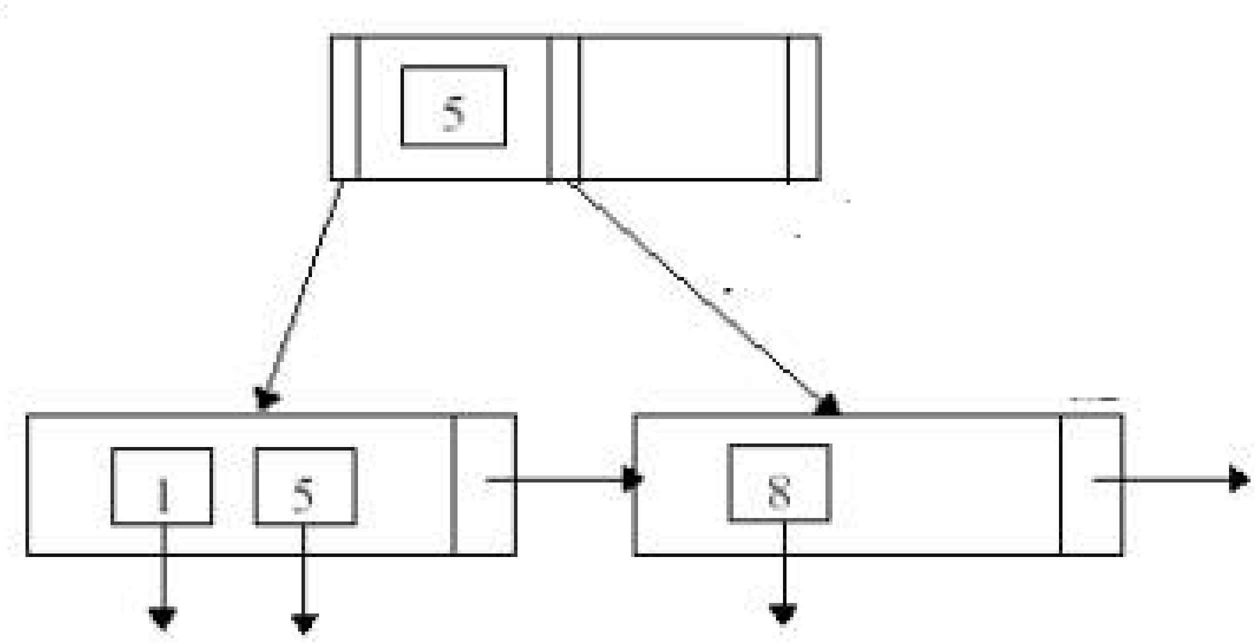
B+- Tree Insertion Cont.

- Inserting value 8
- Since the node has room, we insert the new value.



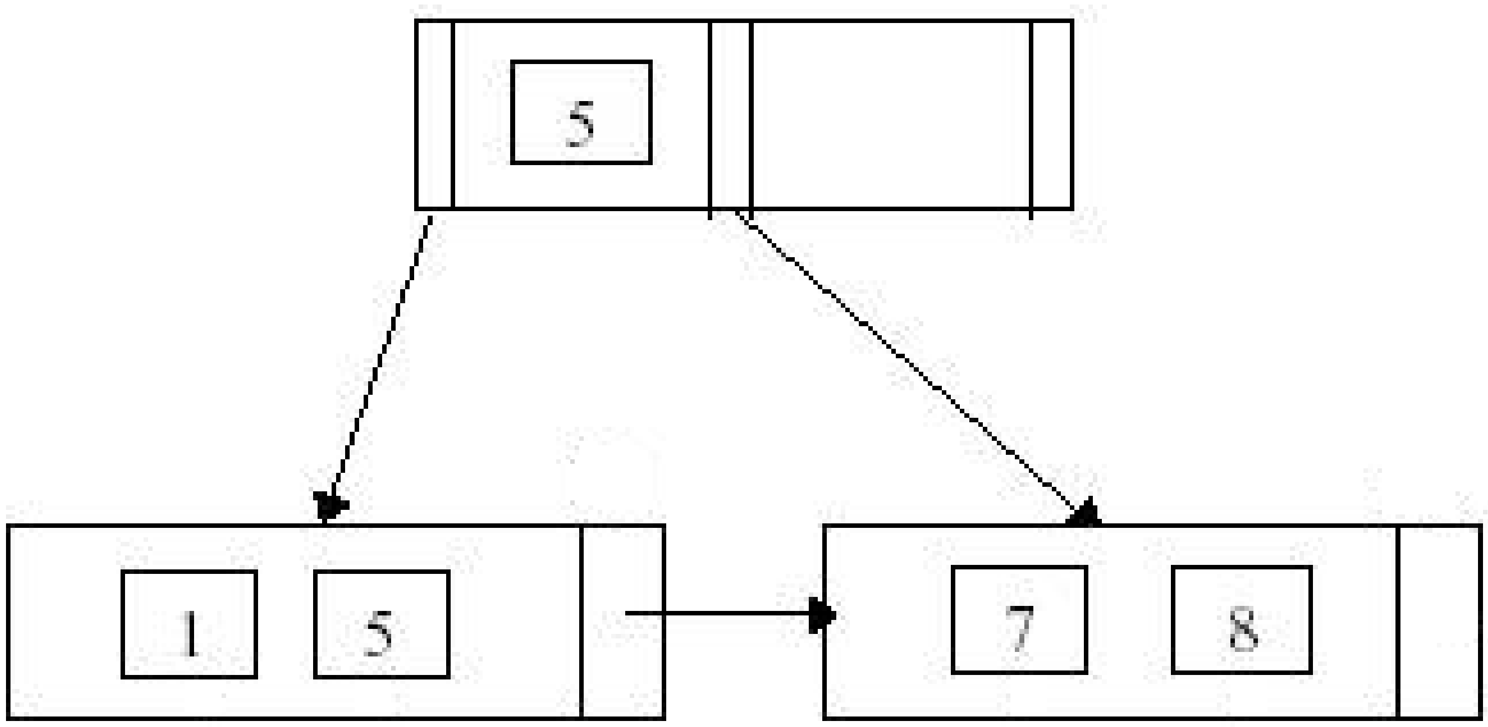
B+- Tree Insertion Cont.

- Insert value 1
- Since the node is full, it must be split into two nodes.
- Each node is half full.



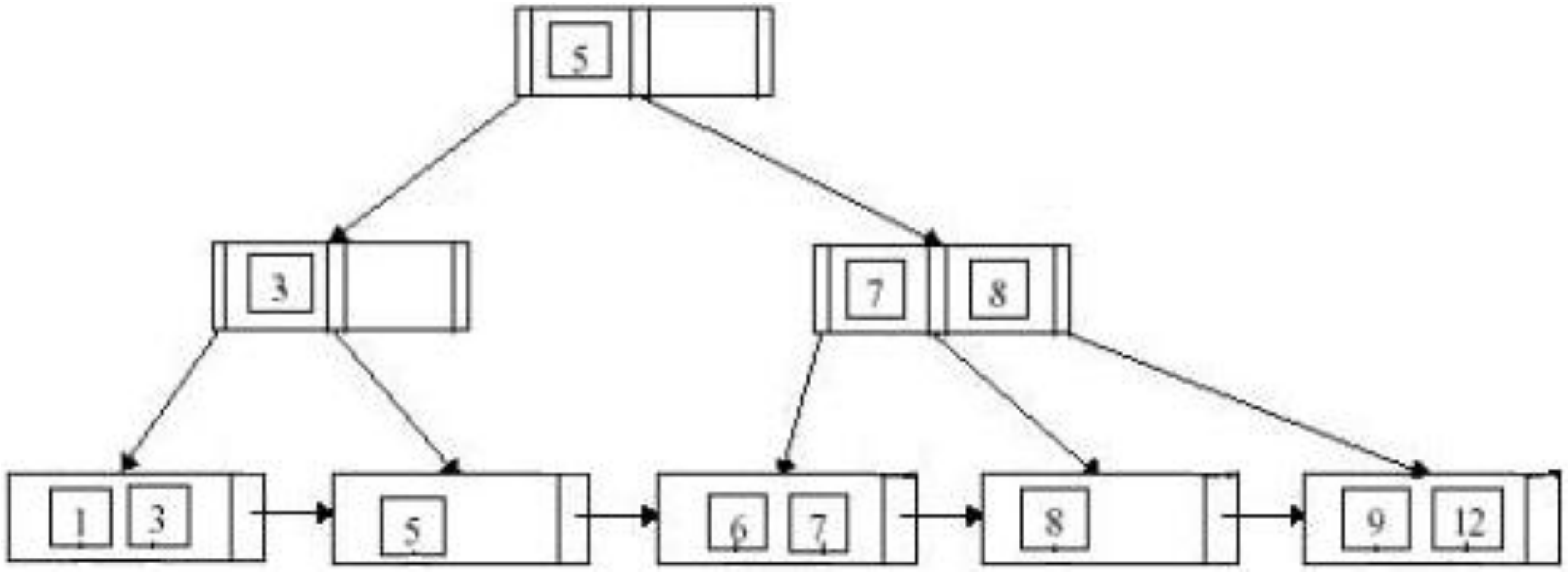
B+- Tree Insertion Cont.

- Inserting value 7.



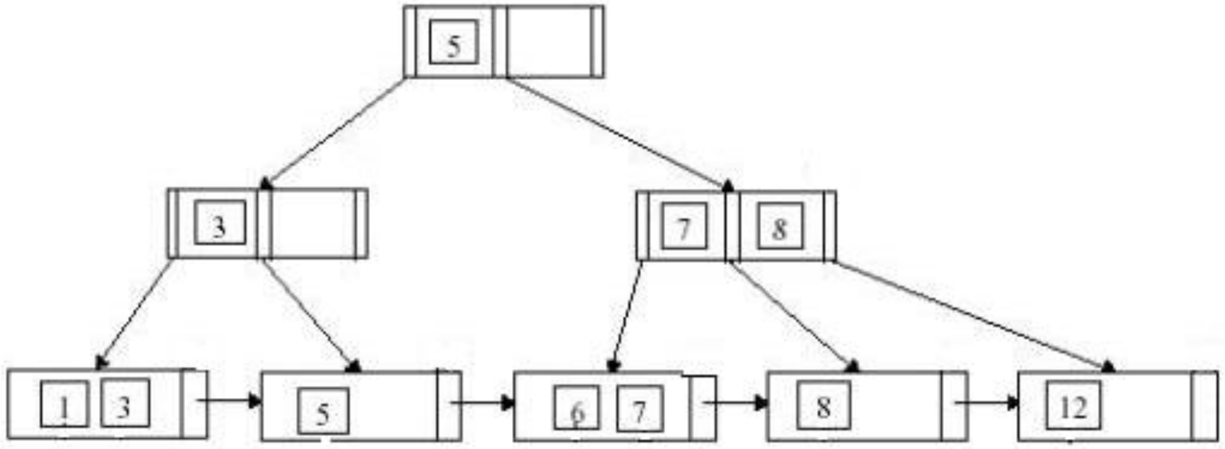
B+- Tree Deletion

- Initial Tree



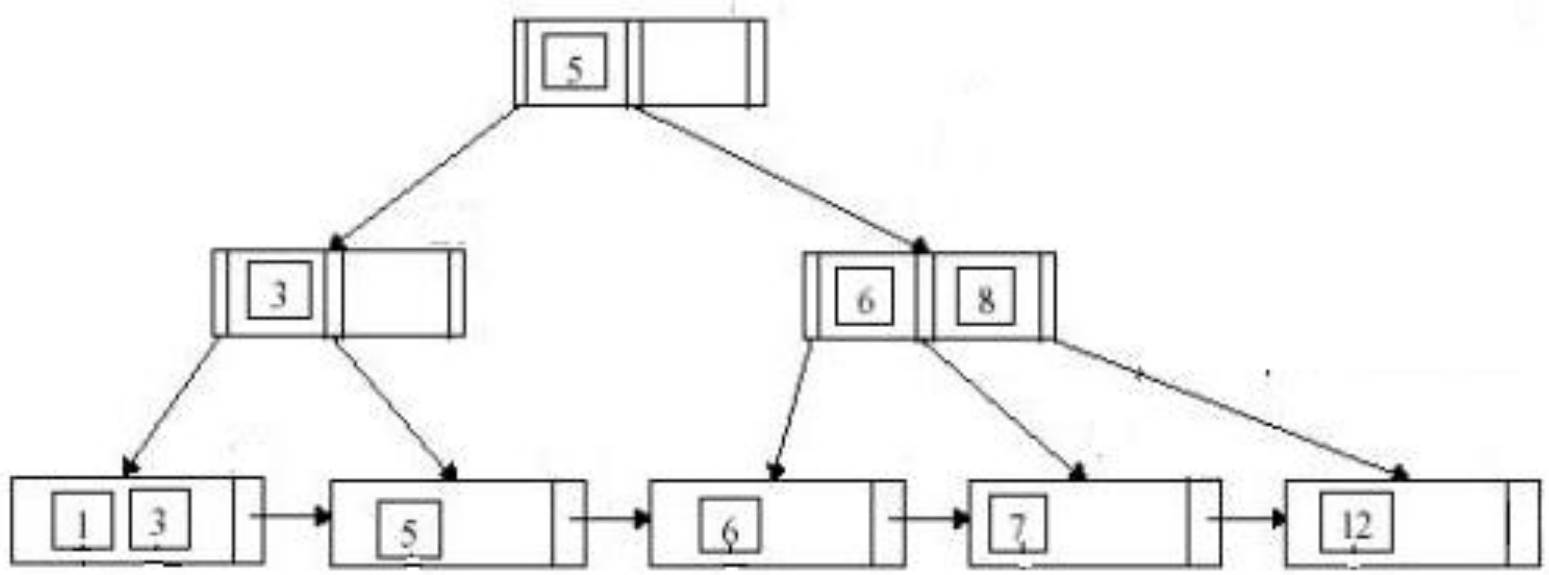
B+- Tree Deletion Cont.

- Delete Value 9
- Since the node is not less than half full, the tree is correct.



B+- Tree Deletion Cont.

- Deleting value 8
- The node is less then half full, the values are redistributed from the node on the left because it is full.
- The parent node is adjusted to reflect the change.



5
3
6
8
1
3
5
6
7
12