Introduction To Multi-way Search Trees

By Yash Gupta

B-Tree

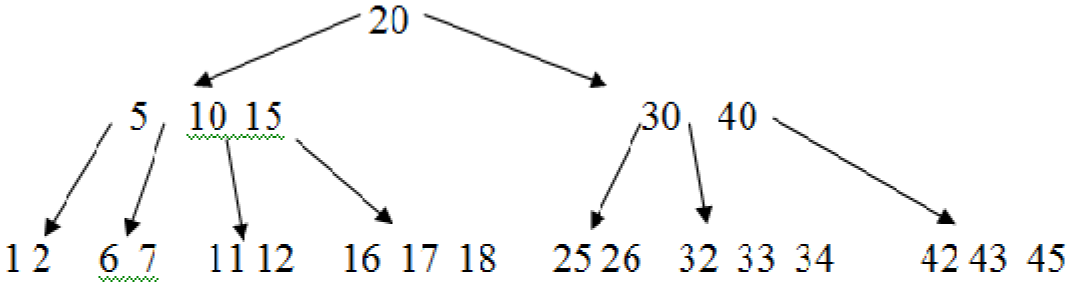
- A B-Tree is a special case of m-way search tree which is balanced having following props
  - The root is either a leaf or it has 2…m subtrees
  - All internal nodes have at least [m/2] nonnull subtrees and at most m nonnull subtrees
  - All leaf nodes are at same level that is tree is perfectly balanced
  - A leaf node has at least [m/2]-1 and at most m-1 entries

B-Tree Insertion

- Whenever we insert a node into B-Tree the insertion takes place in leaf node
- If the resulting insertion causes overflow then we spilt the node and the median key is shifted upwards in parent node and process continues as long as there is no more overflow
- Example – 92,24,6,7,11,8,22,4,5,16,19,20,78

B-Tree Deletion

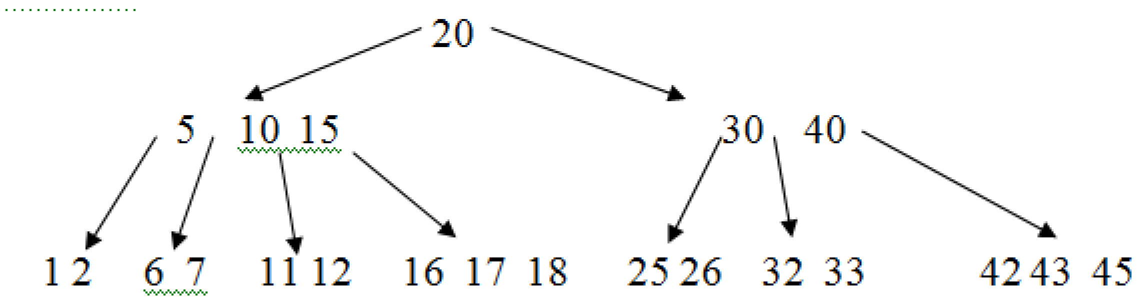
- Assume B-Tree of order 5



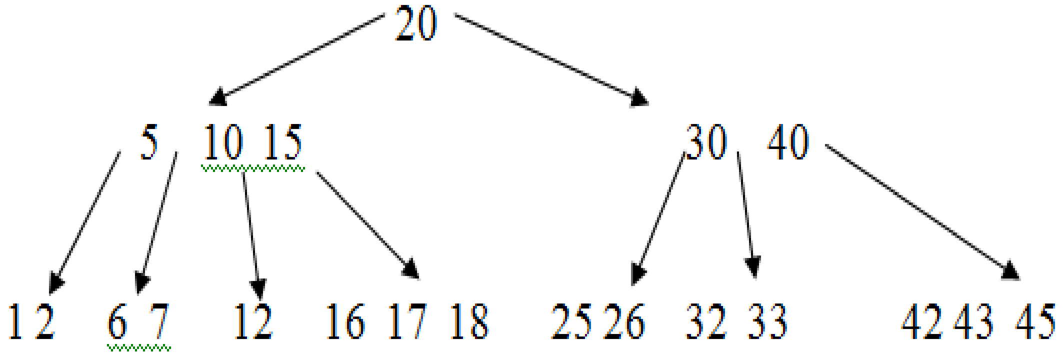
- In order to delete following properties must be maintained
  - The root is either a leaf or it has 2…m subtrees
  - All internal nodes have at least [m/2] nonnull subtrees and at most m nonnull subtrees. i.e 5/2 = 2.5 ~ 3
  - A leaf node has at least [m/2]-1 and at most m-1 entries i.e min [5/2] - 1 =2 entries and max 4 entries

- In order to perform deletion and maintain the properties of B-Tree we need two algorithms
  - Merge : We combine the siblings whenever underflow
  - Borrow : We borrow a node from neighbour whenever there is underflow
- We first look for borrow and if not possible only then we merge

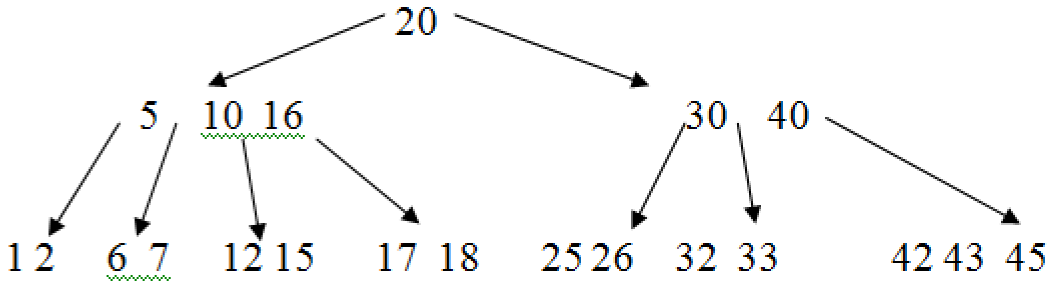
- Delete 34 – The resulting tree is not disturbed



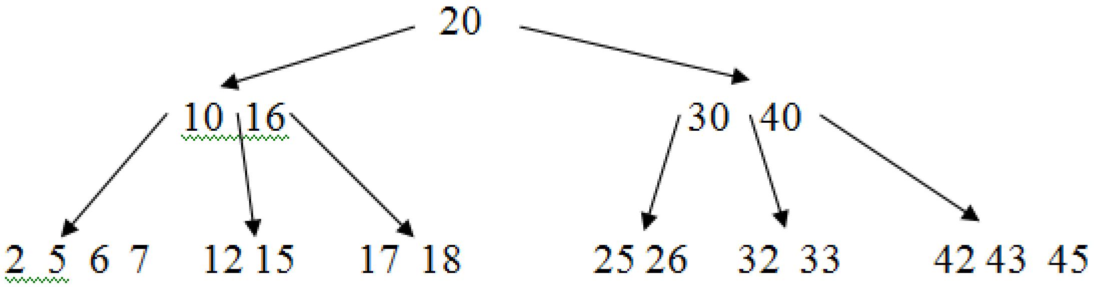
- Delete 11 : It causes underflow (minimum entries must be 2



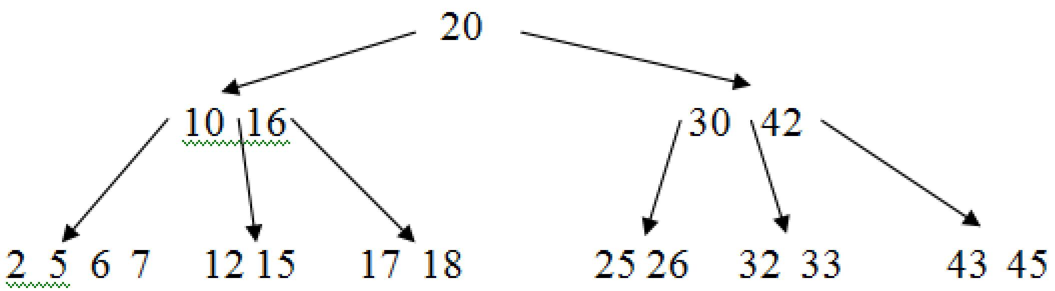
- Borrow– 15 becomes neighbour of 12 and 16 is shifted upwards



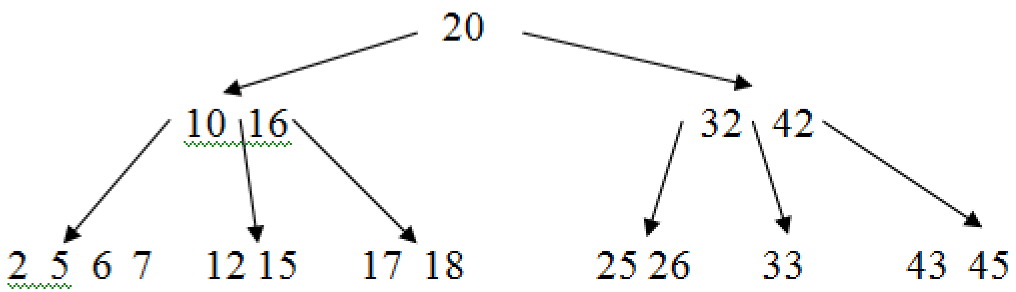
- Delete 1 – It causes underflow and if we try to borrow from neighbouring sibling the sibling becomes underflow hence merge the two siblings



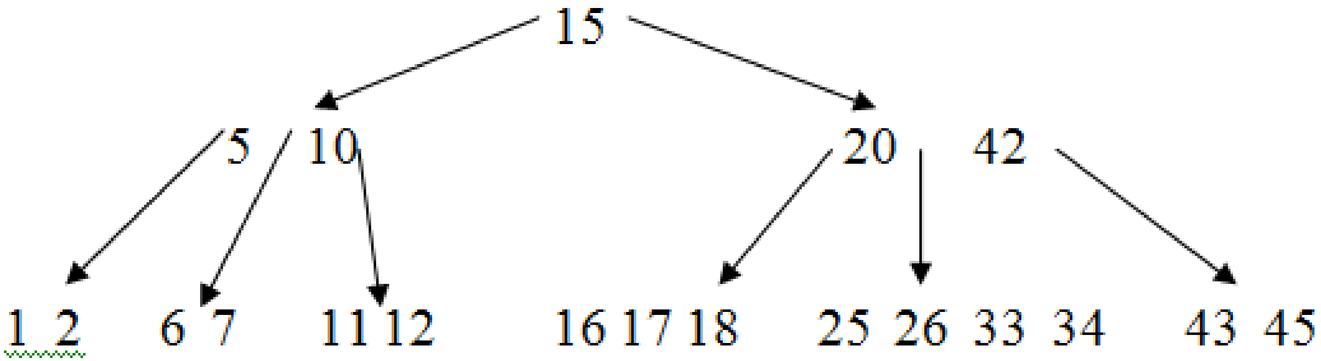
- Delete 40 – Results in underflow



- Delete 30 – Results in underflow borrow 32



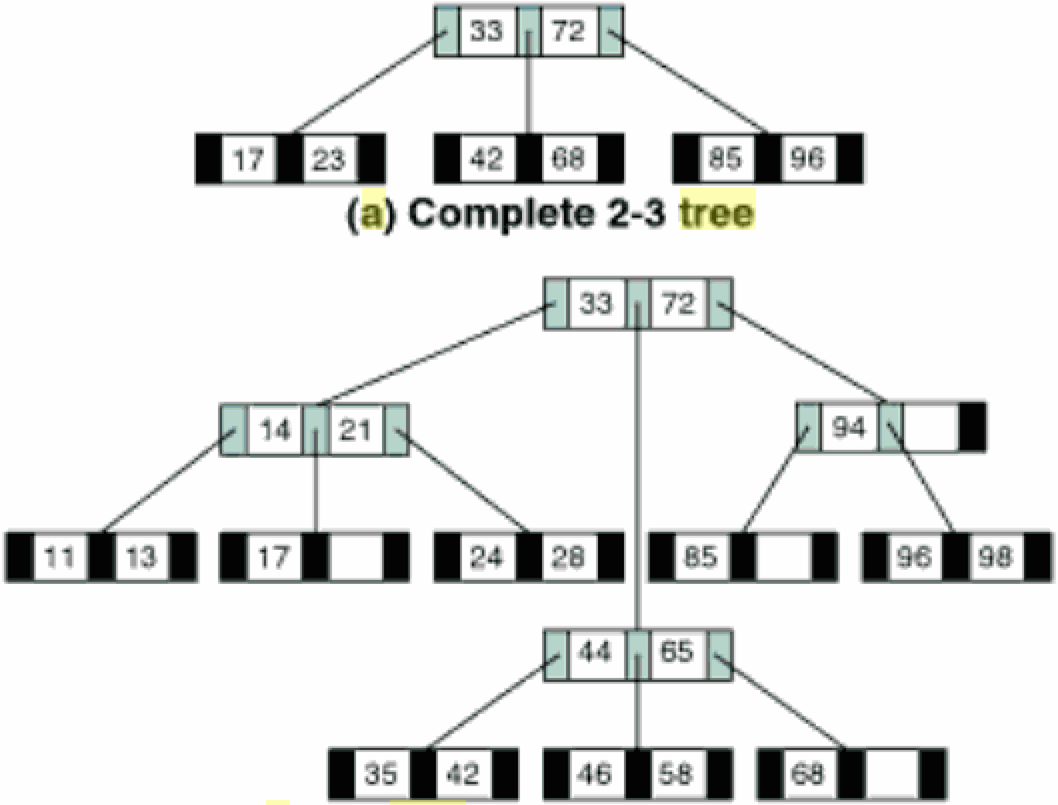
- The node 33 is in underflow we neither can borrow nor merge we need to perform restructuring


15
5 10
20 42
1 2
6 7
11 12
16 17 18
25 26 33 34
43 45

Simplified B-Trees

- 2-3 TREE
- 2-3-4 TREE
- B* TREE
- B+ TREE

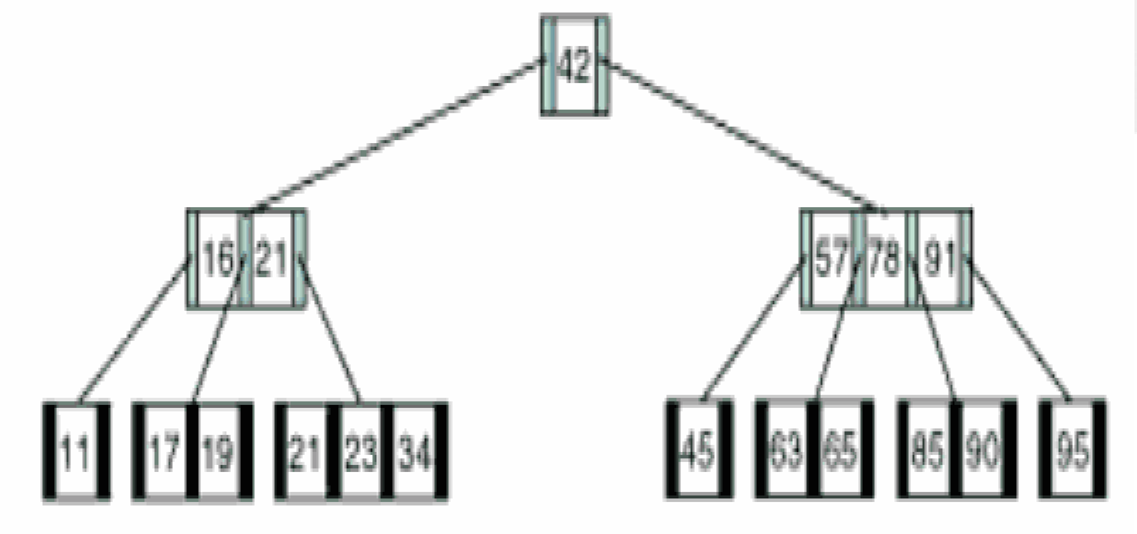
- 2-3 TREE
  - B-TREE of order 3
  - It is named so because each node has either 2 or 3 subtrees



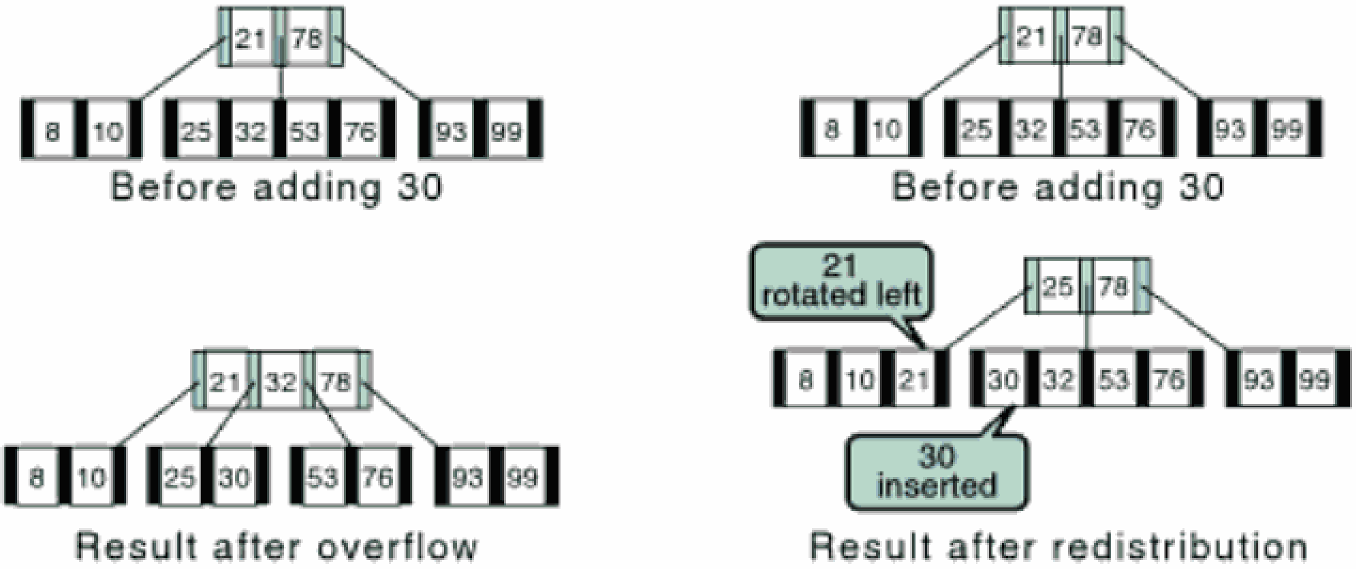
(a) Complete 2-3 tree

(b) 2-3 tree with empty entries

- 2-3-4 TREE
  - B-TREE of order 4
  - It is named so because each node has either 2,3 or 4 subtrees



- B* TREE
  - B-TREE can be space consuming because upto 50% of entries can be empty
  - In this tree instead of node being splitted immediately, the data are redistributed among node’s siblings delaying the creation of new node splitting



(a) Insertion into B-tree of order 5

(b) Insertion into B*tree of order 5

- B+ TREE
  - In large file system data need to be processed both randomly and sequentially
  - B-Tree is time consuming due to moving up-down
  - The B* tree have following difference
  - Each data must be represented at leaf level even though there may be internal nodes with same keys. The internal nodes are used only for searching they do not contain data
  - Each leaf node has one additional pointer, which points to next node in sequence

Lexical Search Tree

- Instead of searching the entire key value, we can consider the key to be a sequence of alphanumeric characters
- For eg, if a key can contain complete alphabet it is known as 26-ary tree
- If a key has 3 letters it has at least 3 levels, If a key has 10 letters it has 10 levels

Tries

- With lexical m-ary tree the tree becomes very large after few levels.
- Tries is a modified form of lexical trees in which those branches are pruned which are not required

Contact Info

- trainers@finaldesk.com
- rishabh@finaldesk.com
- nilesh@finaldesk.com
- jignesh@finaldesk.com
- yash@finaldesk.com
- anand@finaldesk.com