2.1. Insertion Into a B-Tree

To insert value X into a B-tree, there are 3 steps:

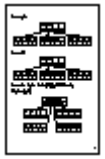
1. using the SEARCH procedure for M-way trees (described above) find the leaf node to which X should be added.
2. add X to this node in the appropriate place among the values already there. Being a leaf node there are no subtrees to worry about.
3. if there are M-1 or fewer values in the node after adding X, then we are finished.

   If there are M nodes after adding X, we say the node has *overflowed*. To repair this, we split the node into three parts:

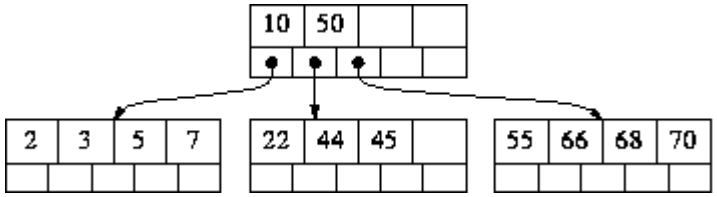
   Left:
   the first (M-1)/2 values
   Middle:
   the middle value (position 1+((M-1)/2)
   Right:
   the last (M-1)/2 values

   Notice that Left and Right have just enough values to be made into individual nodes. That's what we do... they become the left and right children of Middle, which we add in the appropriate place in this node's parent.

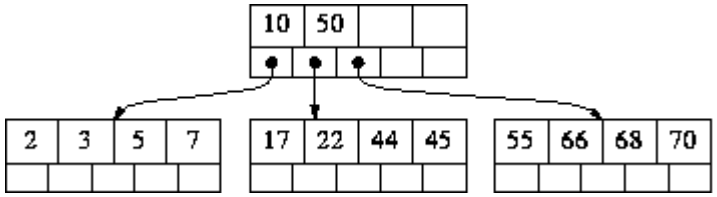
   But what if there is no room in the parent? If it overflows we do the same thing again: split it into Left-Middle-Right, make Left and Right into new nodes and add Middle (with Left and Right as its children) to the node above. We continue doing this until no overflow occurs, or until the root itself overflows. If the root overflows, we split it, as usual, and create a new root node with Middle as its only value and Left and Right as its children (as usual).



For example, let's do a sequence of insertions into this B-tree (M=5, so each node other than the root must contain between 2 and 4 values):



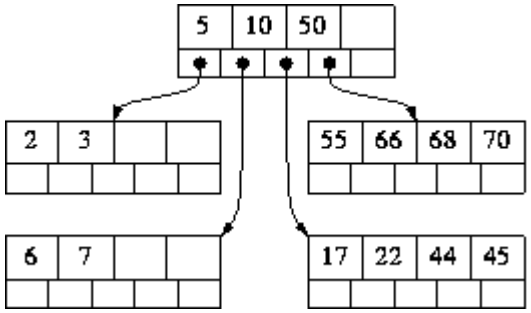
Insert 17: Add it to the middle leaf. No overflow, so we're done.



Insert 6: Add it to the leftmost leaf. That overflows, so we split it:

- Left = [2 3]
- Middle = 5
- Right = [6 7]

Left and Right become nodes; Middle is added to the node above with Left and Right as its children.

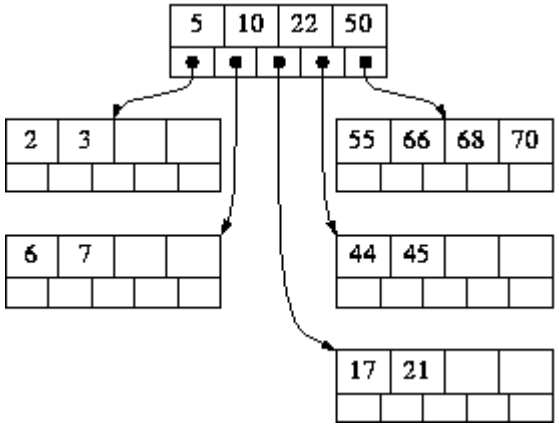


The node above (the root in this small example) does not overflow, so we are done.

Insert 21: Add it to the middle leaf. That overflows, so we split it:

- left = [17 21]
- Middle = 22
- Right = [44 45]

Left and Right become nodes; Middle is added to the node above with Left and Right as its children.

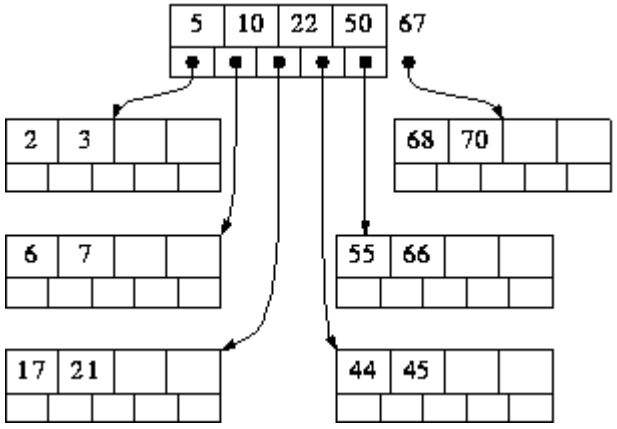


The node above (the root in this small example) does not overflow, so we are done.

Insert 67: Add it to the rightmost leaf. That overflows, so we split it:

- Left = [55 66]
- Middle = 67
- Right = [68 70]

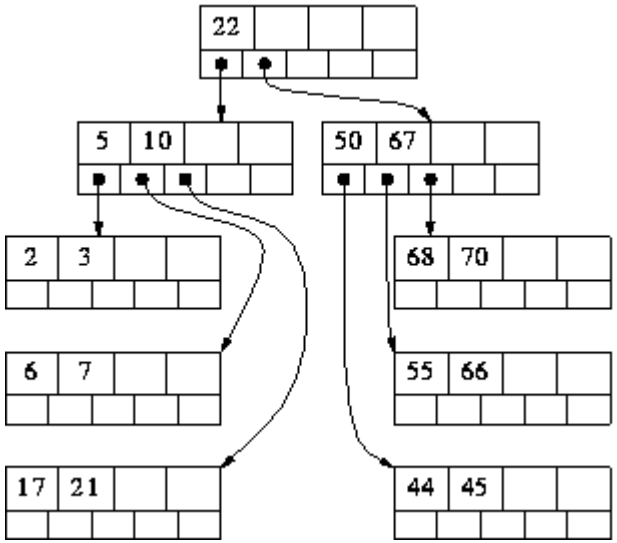
Left and Right become nodes; Middle is added to the node above with Left and Right as its children.



But now the node above does overflow. So it is split in exactly the same manner:

- Left = [5 10] (along with their children)
- Middle = 22
- Right = [50 67] (along with their children)

Left and Right become nodes, the children of Middle. If this were not the root, Middle would be added to the node above and the process repeated. If there is no node above, as in this example, a new root is created with Middle as its only value.



The tree-insertion algorithms we're previously seen add new nodes at the bottom of the tree, and then have to worry about whether doing so creates an imbalance. The B-tree insertion algorithm is just the opposite: it adds new nodes at the *top* of the tree (a new node is allocated only when the root splits). B-trees grow at the root, not at the leaves. Because of this, there is never any doubt that the tree is always perfectly height balanced: when a new node is added, all existing nodes become one level deeper in the tree.