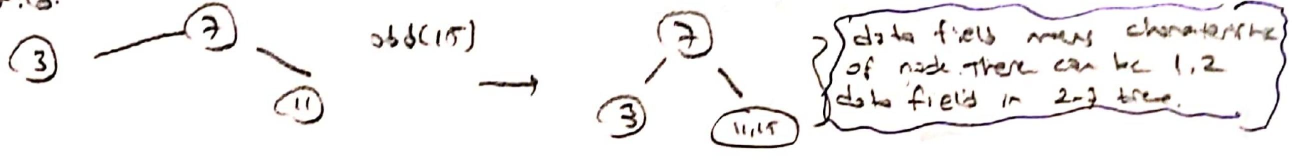
2-3 Tree

insertion rule: A 2-3 tree maintains balance by being built from the bottom-up, not the top down. Instead of hanging a new Node onto leaf, we insert the new node into a leaf, we search the insertion node using the normal process of 2-3 tree.

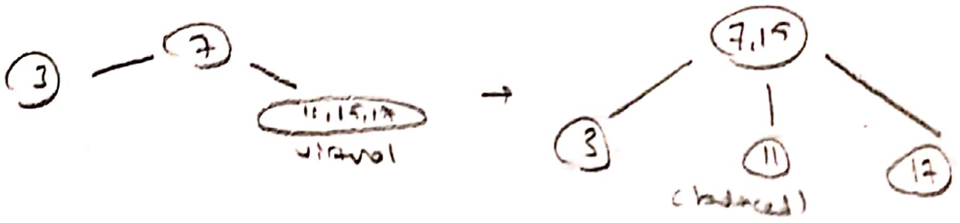
* There are some special cases which defined in the book. while constructing tree, these one will be really helpful.

case1 (inserting into 2-Node Leaf)
* we can insert given node to directly as a data field of appropriate right or left child.

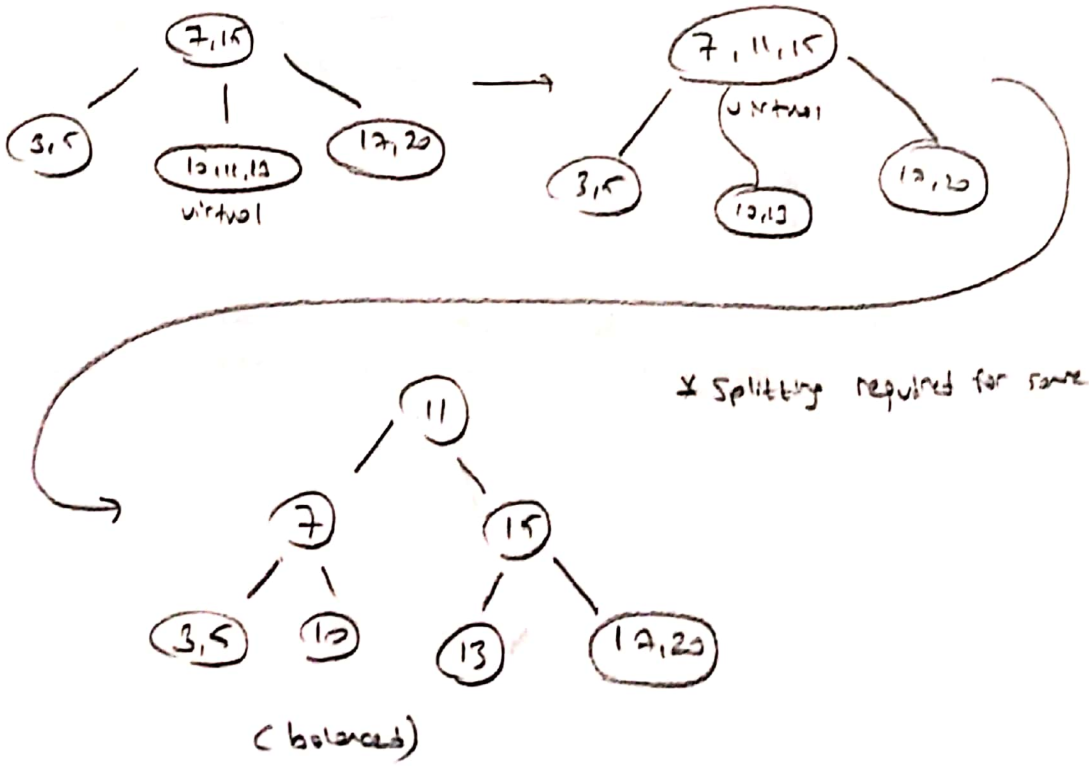


7
3
11
add(15)
7
3
11,15
data fields means characteristic of node. There can be 1,2 data field in 2-3 tree.

case 2 (inserting into 3-Node Leaf with a 2-Node parent)
* Firstly we insert given element to appropriate position. But node cannot keeps 3 values in it, middle value propagate up to 2-Node parent. Then we need to split executed node as 2-node



case 3 (inserting into 3-Node leaf with 3-Node parent)
* All leaf nodes are full, so we insert any other number, one of the leaf nodes will need to be virtually split, and its middle value will propagate to the parent. Because the parent is already a tree node, it will also need to be split.



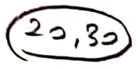
* Splitting required for some nodes.

input = { 20, 30, 8, 47, 39, 18, $\underbrace{40, 24}$ }
↳ last four digit of my numbers.

add (20):

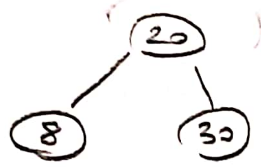
20 * tree is empty, initialize 3 node tree and put 20 as a data field.

add (30)



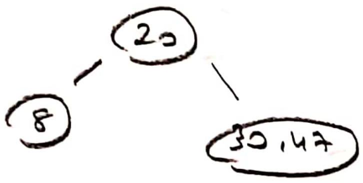
* root was 3-node and 30 was putted as a second data field

add (8)



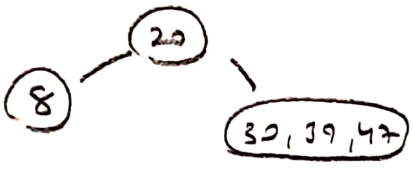
* 8 is less than 20 and we need to split up the tree as 2-Node tree.

add (47)

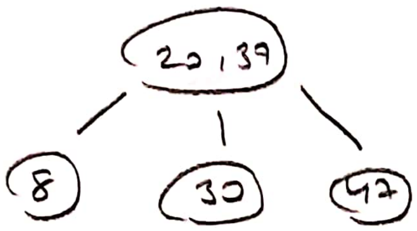


* 47 is greater than 30, and it can be representing as right data field of 3-Node

add (39)

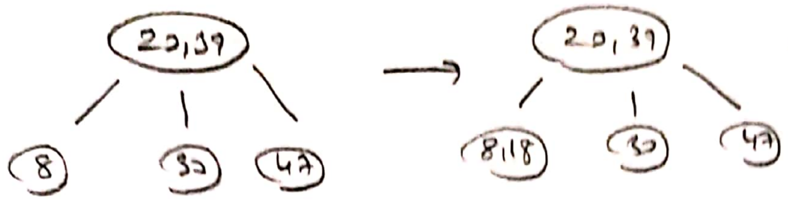


* According to case 2 we need to move middle value to one degree top of current node. And split the current node as a 2-Node.



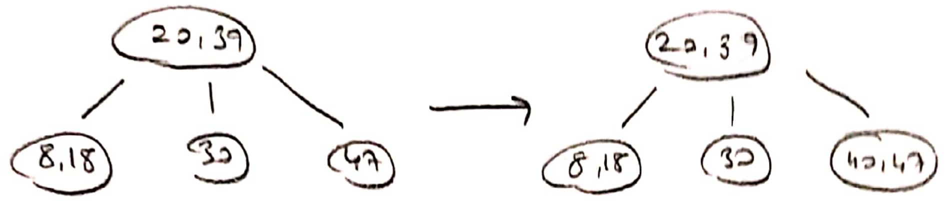
* Now root node became 3-Node and it has to be 3 different child so right most of tree splitted.

add(18)



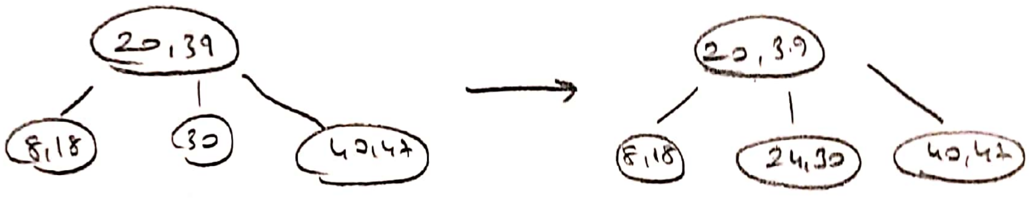
* According to de facto binary-search-tree rule 18 can be putted as a data field of left most appropriate node.

add(40)



Same rule is valid for "40"

add(24)



Final Tree

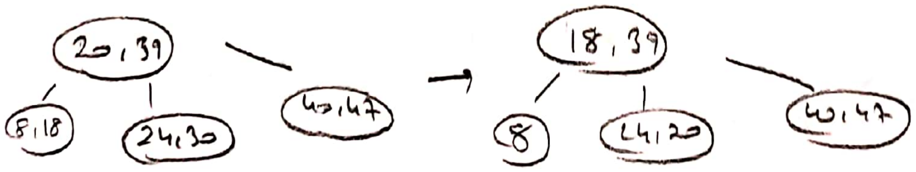
Same rule is valid for middle value it can be assigned as value left data field of middle node.

* Removing from 2-3 tree is somewhat the reverse of the insertion process. To remove an item we must search for it.

→ if the item to be removed is in a leaf, we simply delete it.

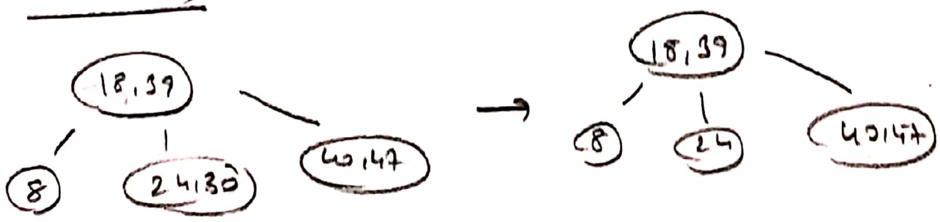
→ if the item to be removed is not a leaf, we remove it by swapping with its inorder predecessor.

remove(20)



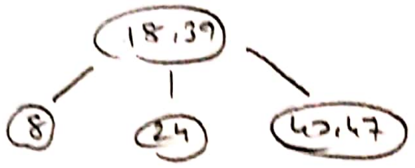
* 18 is the in order predecessor of 20. It move to the top.

remove(30)

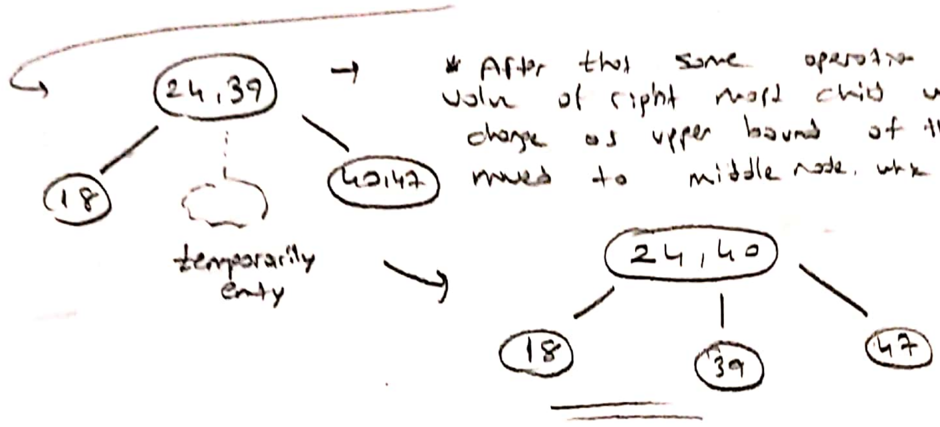


* 30 is a leaf node data field and it can be removed directly.

remove (8)

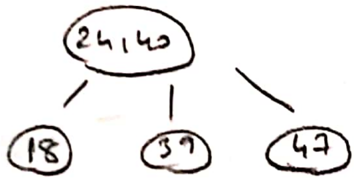


* Removing 8 will cause some swap operations.
* 18 is not lower band of root any more and it is needed to be change as lower of middle node After that new lower band is 18. It can substitute of removed node data field

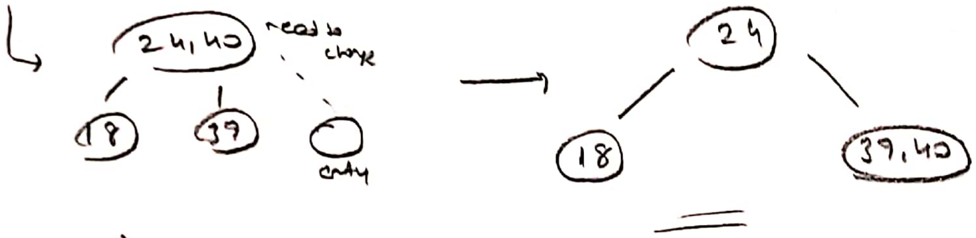


* After that same operation can be applied for lower valu of right most child which is 40. It can be change as upper bound of the root node. 39 can be moved to middle node. which is enty.

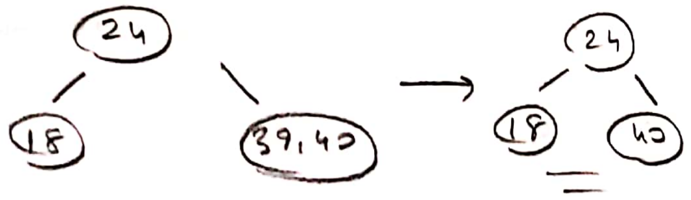
remove (47)



* After that removing right-most node will be enty and root can not be 3-Node any more because 4 data will be remain for constructing. We can make root node 2-Node then middle node can be used to keep removed element from. root node. left most node also could be chose but, I choose middle one.

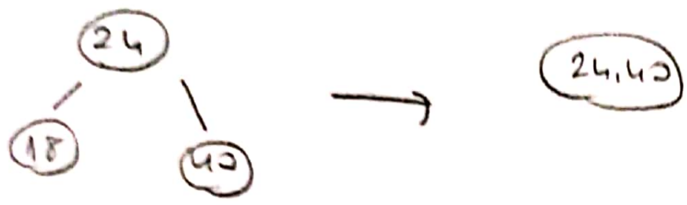


remove (39)



* It can be directly removed because, it was located in a leaf node.

remove (18)



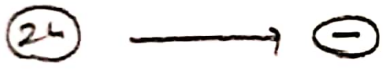
* After removing "18", there will not be enough data to constructing 2-node tree
* we can also remove right most tree value and add to root for constructing 3-Node.

* Our purpose is keeping the sub-trees height same. By doing that our search will be O(logn) obvious

remove (40)

* Just remove data in the root node.

remove (24)



* All elements are removed one by one first-in first-out rule.

* Advantage of 2-3 tree:

1) It has fewer complicated operation according to AVL and Red-Black tree.
2) There were no rotation.
3) we say that the time complexity of 2-3 tree is O(logn) with no doubt.