2-3 Deletion (Case #1)

This lesson will cover the deletion operation in 2-3 trees, discussing all the four scenarios based on the position of the element which needs to be deleted.

We'll cover the following

Deletion Algorithm:#

Deletion in 2-3 Trees is implemented based on the same scenarios we discussed in the last lesson but in the reverse order. Deletion algorithm also takes $O(Logn)$ time. And just like insertion, deletion also begins from the leaf node. The deletion in 2-3 Trees is performed based on these scenarios:

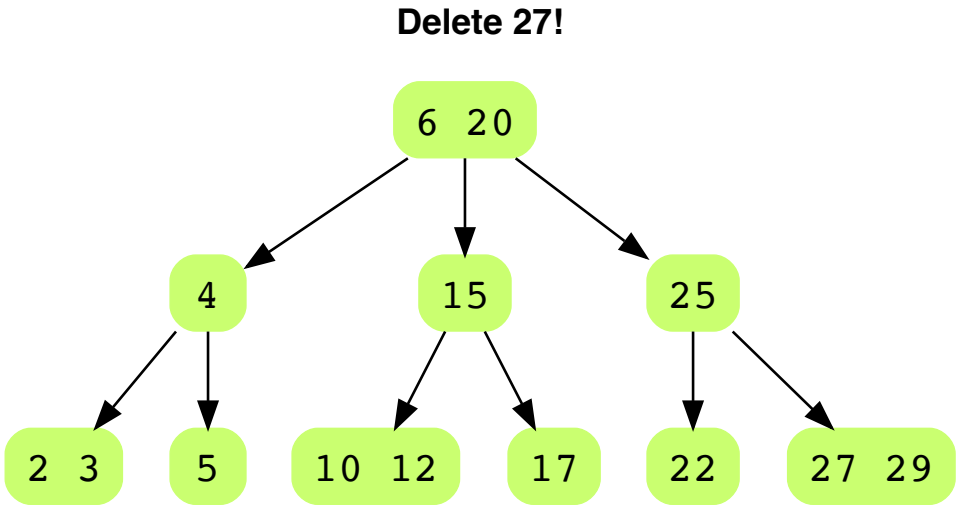
Case 1: Element at Leaf:#

When the element which needs to be removed is present at t[illegible] w check how many keys are present in that node; this further divides the algorithm into two scenarios:

1.1 Leaf node has more than one key:#

If the leaf at which the element to be deleted has more than one key, then simply delete the element.

Example: See the following example where the node has more than one keys.



1.2 Leaf node only has one key:#

If the leaf node where the element to be removed is present has only one key, then we will have to adjust the keys of that sub-tree in such a way that it remains ordered and balanced. This condition is further divided into two scenarios:

1.2.1 Any of the siblings has two key

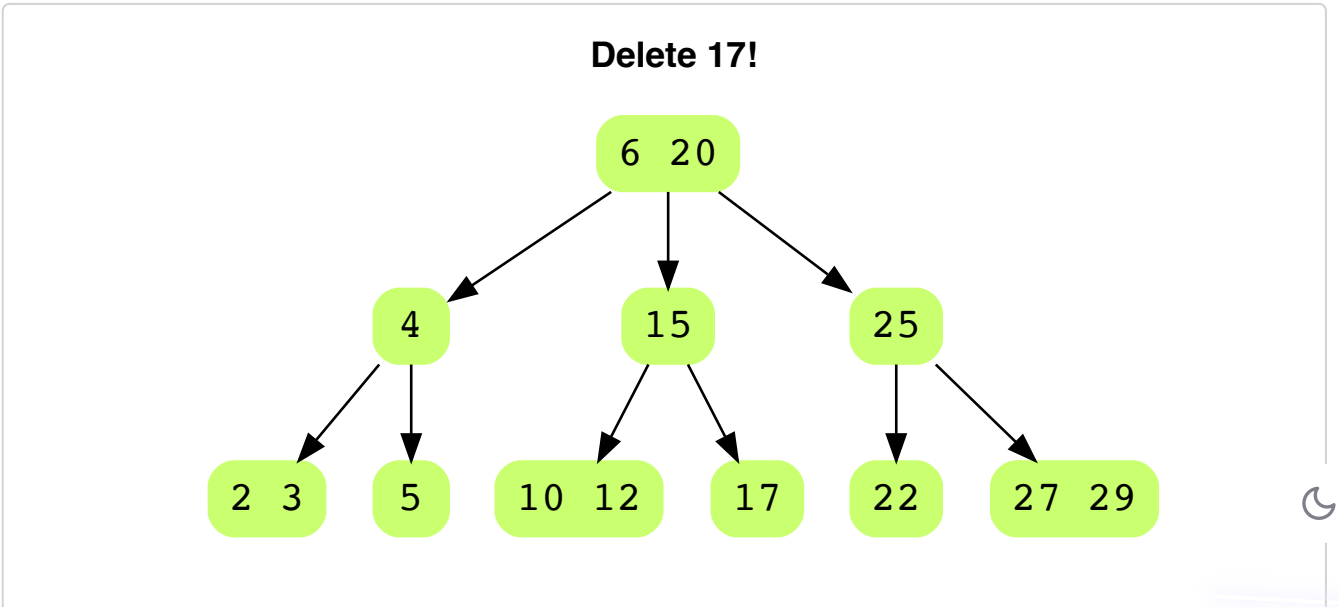
Siblings mean the other adjacent leaf nodes that share the same parent. A node could have one or two siblings depending upon its position. Check how many keys are present at left or right sibling nodes. If any of the siblings have more than one key, then your problem is solved. All you need to do is move an element from the sibling node to the parent node and shift down a node from parent to your node. This process is called *Redistribution by Rotation* and it can be performed in two ways:

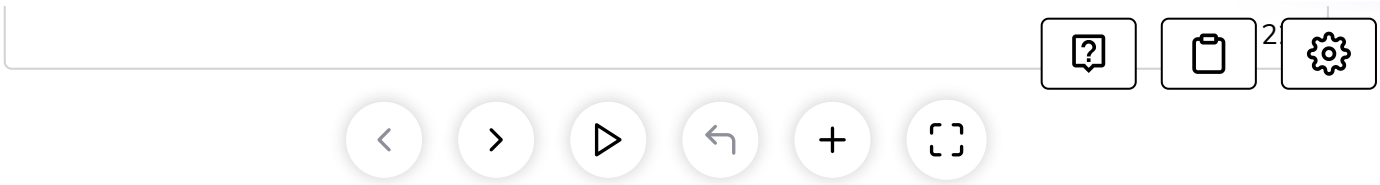
Rotation from Left Sibling:#

In this case, we lend a key from left sibling by shifting up the key having largest value in the node to parent node and move down parent node key (most right) to our node.

Rotation from Right Sibling:#

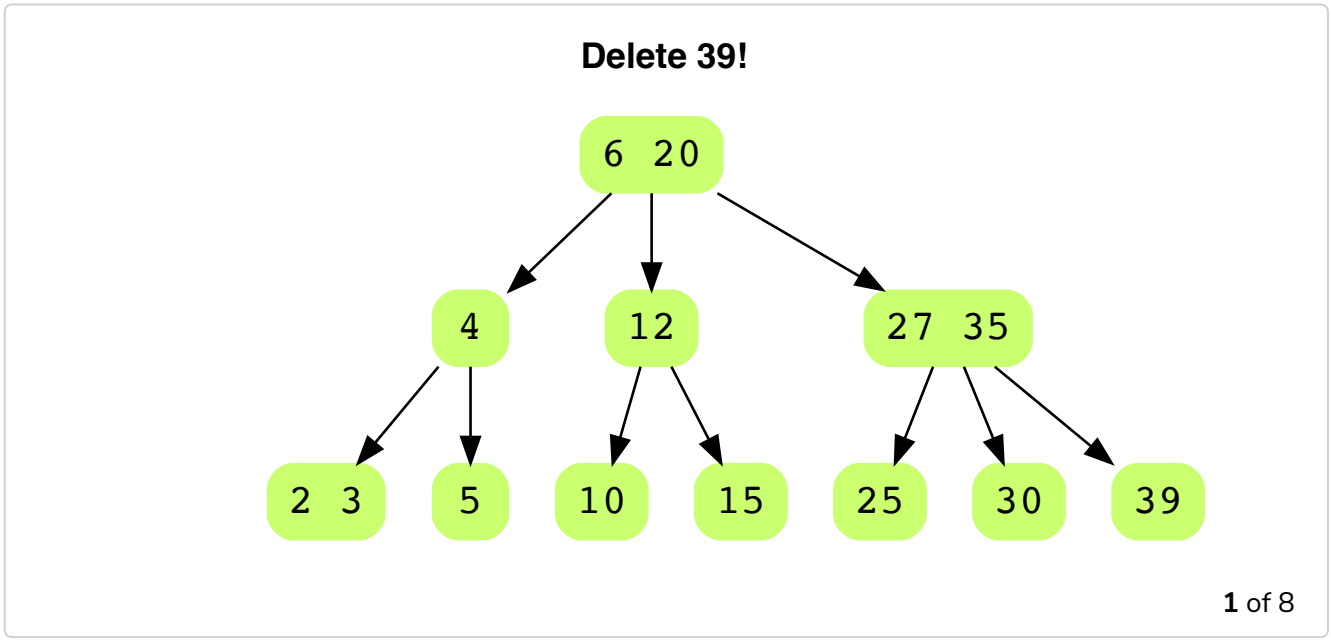
In this case, we lend a key from right sibling by shifting up the key having smallest value in the node to parent node and move down parent node key (most left) to our node.





1.2.2 No sibling has more than one key:

- In the case when none of the siblings have more than one key, we have no other option but to merge the two nodes by rotation of key. So we merge two child nodes into one node by rotating elements accordingly. This process is called *Merge by Rotation*.
- If child nodes have more than one keys, we shift an element from the child node to make it the parent node. When we are left with only one key at each child node, then we are bound to delete the node.



Now we are only left with the last case, i.e., when a node is a Parent node which will be covered in the next lesson.

Back

2-3 Insertion

Next

2-3 Deletion (Case #2)

Mark as Completed

Report an Issue