Deletion from a 2-4 Tree

First we must note that if a non-leaf node value is to be deleted, we can just as easily replace that value with the largest value in its left subtree or the smallest value in its right subtree. (Keep in mind that "left subtree" refers to the subtree DIRECTLY to the left of the value, pictorially speaking. In the text they state to find the "right-most" node in the ith subtree. The ith subtree is the left subtree, and "right-most" means largest.) Thus, we only have to consider the deletion of values from leaf nodes.

In the standard case, a value can simply be removed from a leaf node that stores more than one value and no structural change has to be made to the tree.

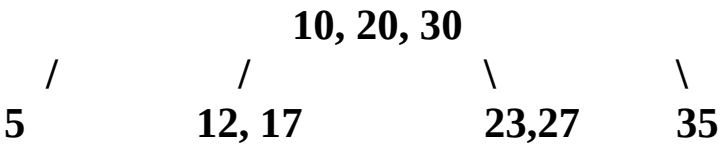
However, it's very possible that the value to be deleted is the ONLY value in the leaf node. The problem with getting rid of this node then is that it would violate the 2-4 tree property that all leaf nodes MUST be on the same height of the tree. In this situation, we will break our work into two cases:

1) An *adjacent* sibling has more than one value stored in its node.

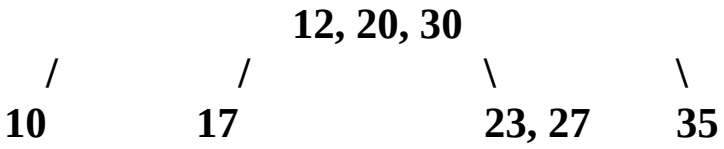
2) An *adjacent* sibling does NOT have more than one value stored in its node, and a fusion operation MUST be performed.

The first situation isn't too difficult to deal with since the necessary changes remain localized to the parent sub-tree of the value to be deleted.

Consider deleting 5 from the following tree:

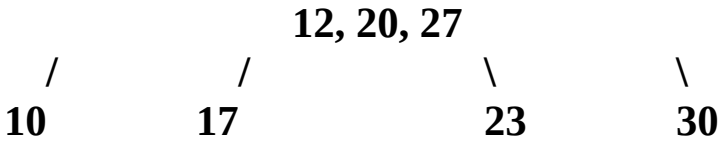


Since the immediate sibling has two values, 12 and 17, we can perform a simple transfer operation:



The idea is to take the 10 to replace the 5, and then simply replace the 10 with the smallest value in its right subtree. This is okay, since there is more than one value at this subtree.

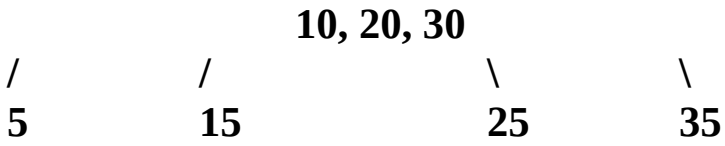
To see the symmetrical case, consider deleting 35 from the tree above. This yields the following tree:



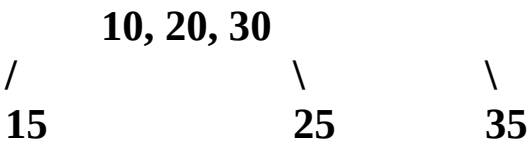
Here, 30 slides down to take the place of 35, and then we must replace the value where 30 used to be with the maximum value from its left subtree.

A fusion operation is a bit more difficult to deal with since it may result in needing another fusion operation at a parent node. But the basic idea of a fusion operation is to reduce the number of siblings of a deleted node. (So here, a node DOES physically get deleted, whereas in the previous case we only shifted values around in existing nodes.)

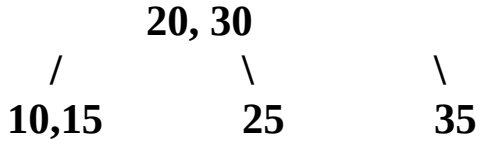
A fusion operation fuses the node to be deleted with a sibling node. Of course the problem with JUST doing this is, if we have less siblings, then our parent stores one too many values. Consider this illustration:



Now, delete 5 and fuse its node with the node for 15:



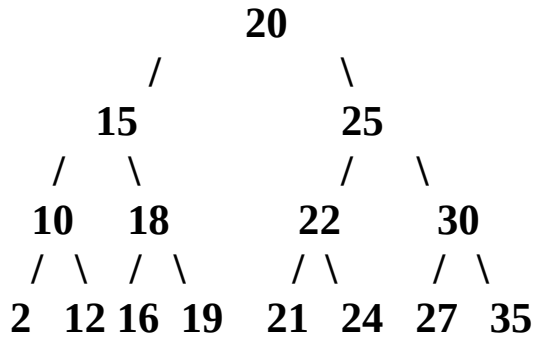
The problem here is that we only have three child nodes when we should have four. The other way to view it is that we have too many values stored at the parent of 5! Thus, to remedy this situation, we can simply drop a value from the parent into the fused child! In this case the value must be 10:



As long as the parent previously contained more than one value this is the end of the operation. (Can you determine which value from the parent node should be dropped into the child in general?)

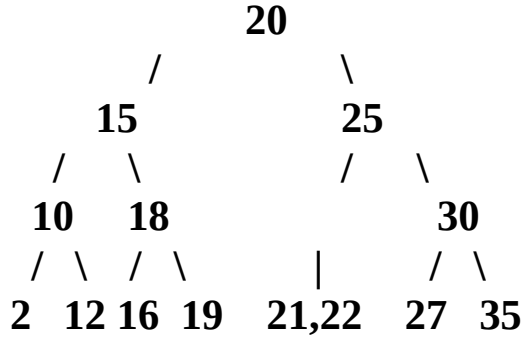
But, we may create a situation where the parent node only has one value and when we drop that value into the child, we create a node with no values in the tree, which is NOT allowed.

Thus, we continue the process of dropping a value from a parent node into an empty child until a 2-4 Tree is formed, OR until we end up dropping the root value into a child. In this case, we'll end up getting rid of the old root and fusing the siblings of that old root into one root node. Consider the following tree:

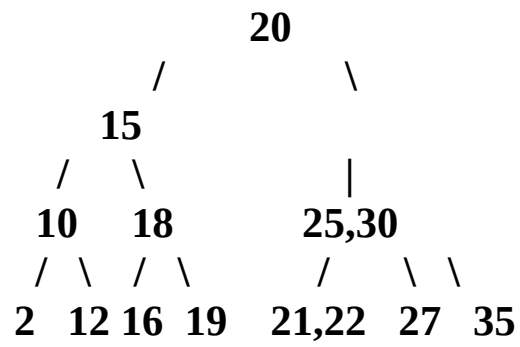


Delete 24:

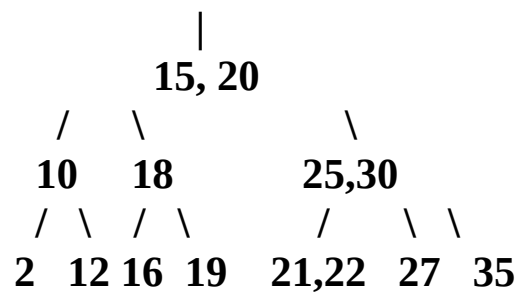
1) Fuse the node with 21 with the one that used to store 24 and drop the parent 22 into that node:



2) To deal with the empty node, drop a value from its parent into the node and perform another fusion operation:



3) But, this creates an empty node at the parent of 25,30. Thus, we must do one more fusion operation. In order to complete the fusion operation, the parent node value 20 needs to be dropped into the child node:



4) Finally, when this occurs, there is NO need for that original root node which now lays empty. So, our final picture is: