Treees – 3

Deletion

- -leaf node
- - node with one child
- - node with two children

Deleting a node from a Binary Search Tree

Deletion of a node is not so straightforward as is the case of insertion. It would depend on which particular node is being deleted.

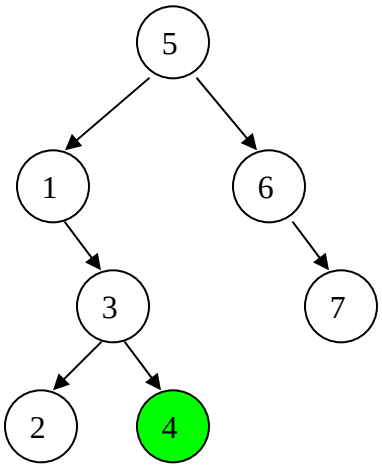
In fact, we note that there can be three separate cases and each case needs to be handled somewhat differently. The various cases are:

(1) deletion of a leaf node,
(2) deletion of an internal node with a single child (either a left or right subtree),
(3) deletion of an internal node with two children (having both left subtree and right subtree.)

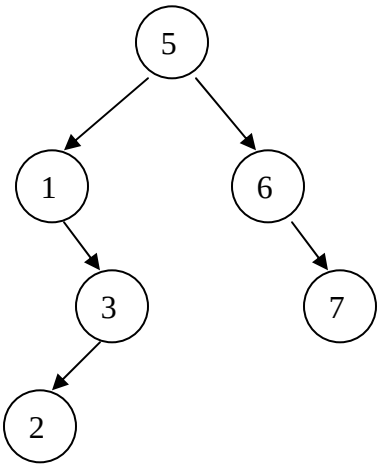
We'll examine each case separately:

Deletion of a leaf Node

Since a leaf node has empty left and right subtrees, deleting a leaf node will render a tree with one less node but which remains a BST. This is illustrated below:



A BST with a leaf node Marked for deletion.

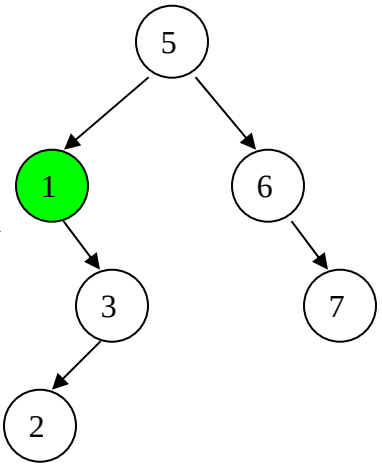


Still a BST

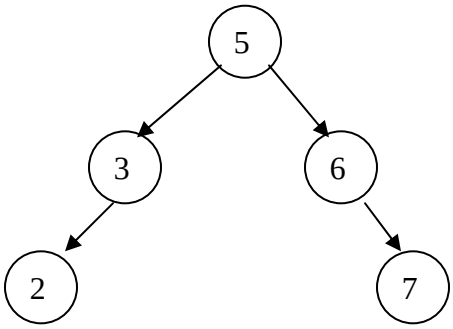
Deletion of a Node with one child

In this case, when the node gets deleted, the parent of the node must point to its left child or its right child, as the case may be.
The parent's reference to the node is reset to refer to the deleted node's child. This has the effect of lifting up the deleted node's children by one level in the tree.
An example is shown below.

A BST with an internal node having only one child marked to be deleted

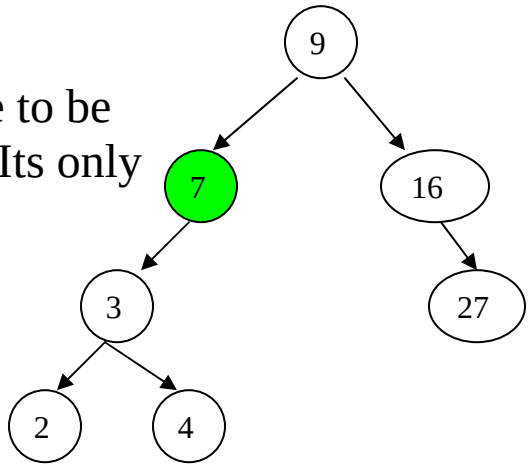


The marked internal node has only a right subtree so the parent of the deleted node will now reference the deleted node's child

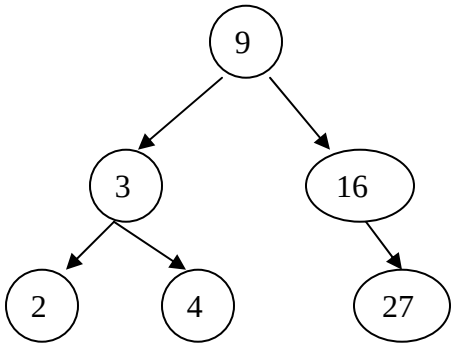


Note that it makes no difference if the node to be deleted has only a left or a right child. The previous example illustrated the case when the only child was a right child. The next example illustrates the case when the only child is a left child.

Initial BST with the node to be deleted shown in green. Its only child is a left child



The BST after the deletion



Deletion of a Node with two child nodes

The last case of deletion from a BST is the most difficult to handle. There is no one-step operation that can be performed since the parent's right or left reference cannot refer to both node's children at the same time.

There are basically two different approaches that can be used to handle this case:

deletion via merging

and

deletion via copying

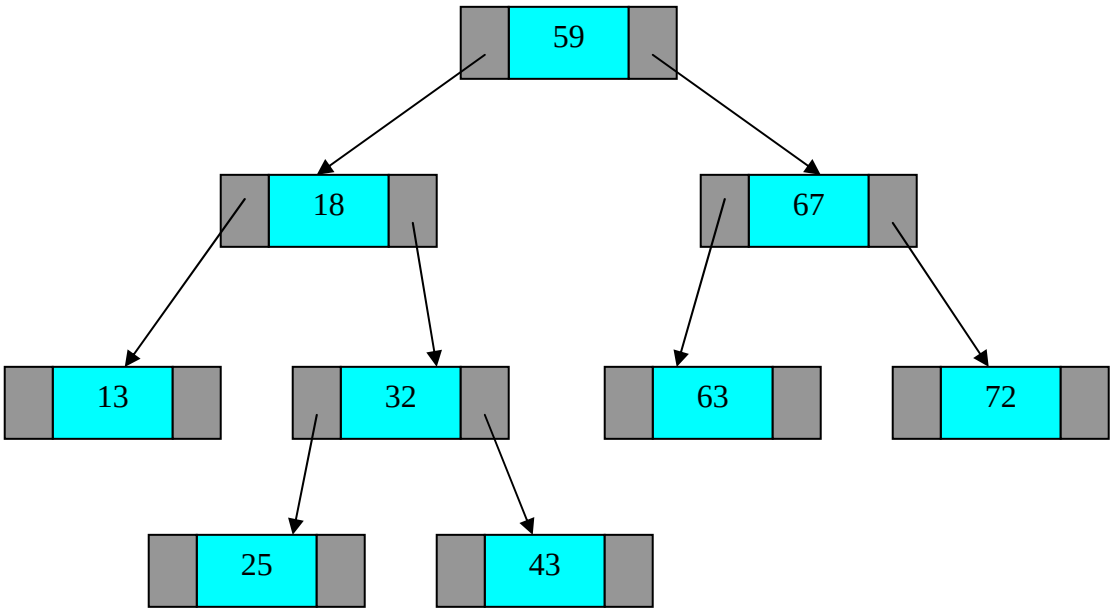
which essentially reduce to the following scenario:

> A deleted node with two children must be replaced by a value which is one of:
>
> - The largest value in the deleted node's left subtree.
> - The smallest value in the deleted node's right subtree.

The above technique means that we need to be able to find, either the immediate predecessor or the immediate

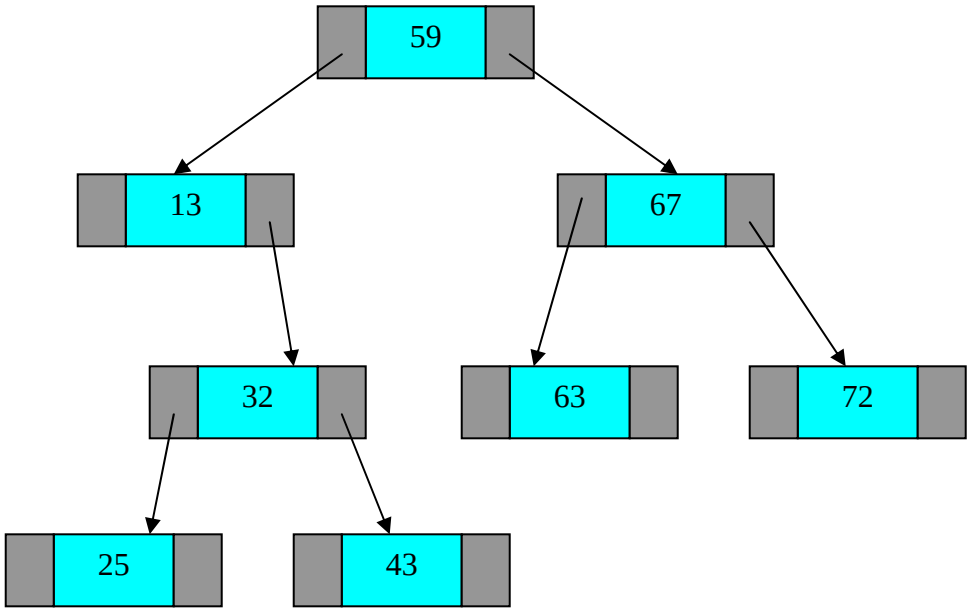
successor node to the node which is being deleted and replace the deleted node with this value.

As an example, consider the following BST and suppose that we are deleting the value 18 from this tree.

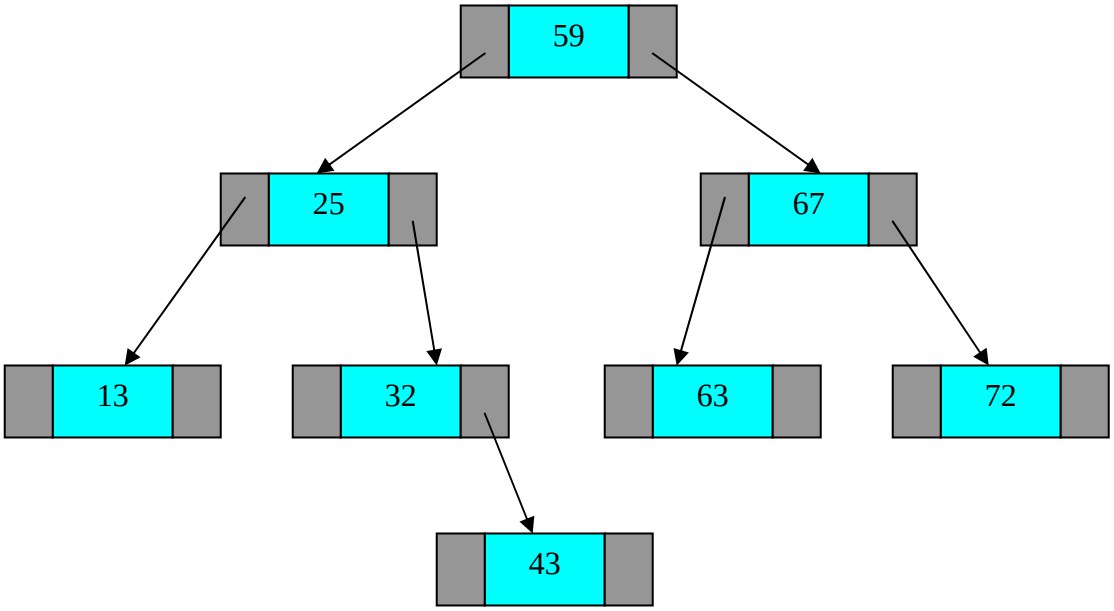


Since the node containing 18 has two children it fits into this category for deletion.

Its immediate predecessor is the rightmost node in its left subtree (which is 13),
so our first choice would be to move 13 into the node currently occupied by 18, this is shown below:



We could have, just as easily, found the immediate successor of 18 which is the leftmost node in its right subtree and put this value into the place currently occupied by 18.
This case is shown below.



Notice that in both cases, the node which is physically deleted from the BST is a leaf node, and this is the trivial deletion case.

Also notice, that while there is no fundamental difference is selecting the immediate predecessor or the immediate successor as the replacement for the deleted value, in reality there may be a difference.

The example above, illustrates, to some degree, this difference which results from a potential difference in the heights of the two subtrees.

In the example above, the immediate predecessor was the better choice since it was only one level away from the node to be deleted and therefore our search to find this node would be shorter than the search to find the immediate successor which was two levels away.

While a few levels difference in the location of the immediate predecessor and immediate successor may not make much difference,

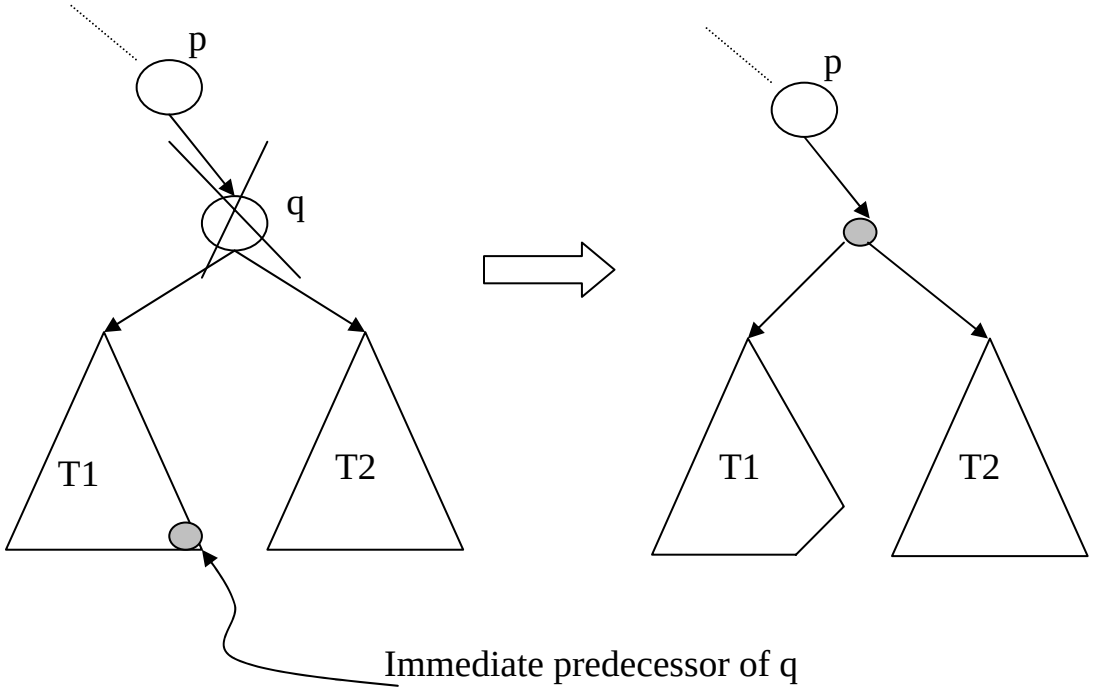
it certainly will if there is a big difference between the two heights and obviously, the shorter the height the quicker the search and this is the way to go.

The General case of Deletion of a node having two child nodes.

In the following tree, we want to delete the node q. The node q has T1 as left subtree and T2 as right subtree.

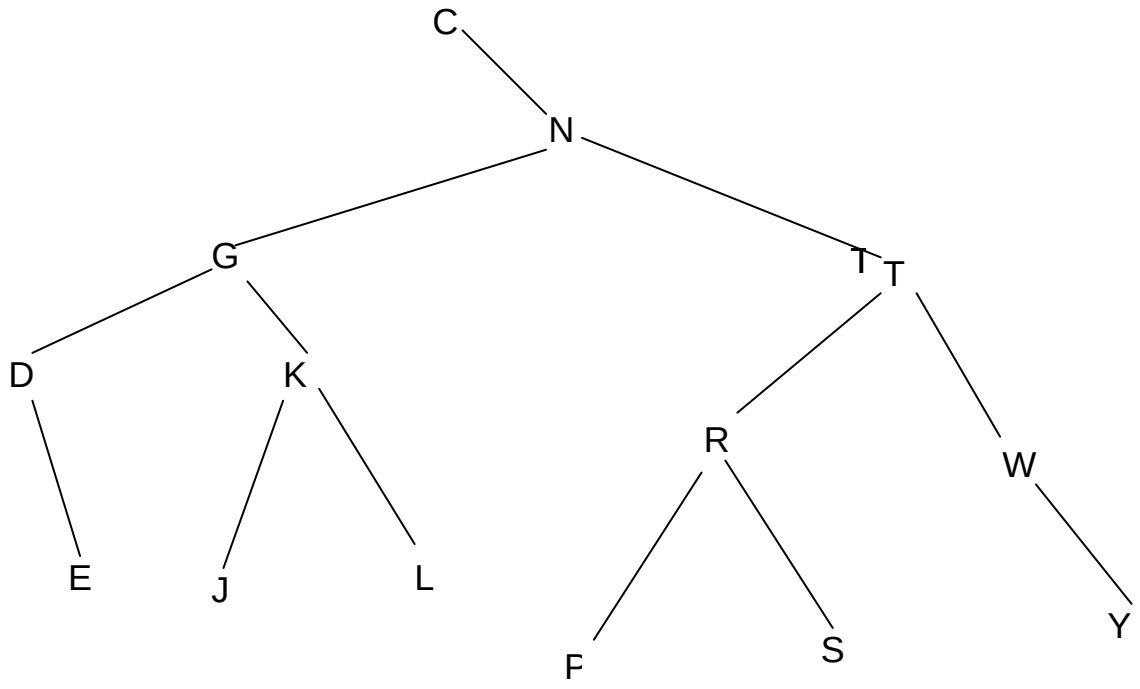
Note that all nodes of T1 are going to be smaller than nodes of T2.
Note further that the rightmost node of T1 will have the largest value in that subtree.

It will be immediate predecessor of node q. All nodes in T2 will be successor of this node.



Let us take a specific example to illustrate the point:

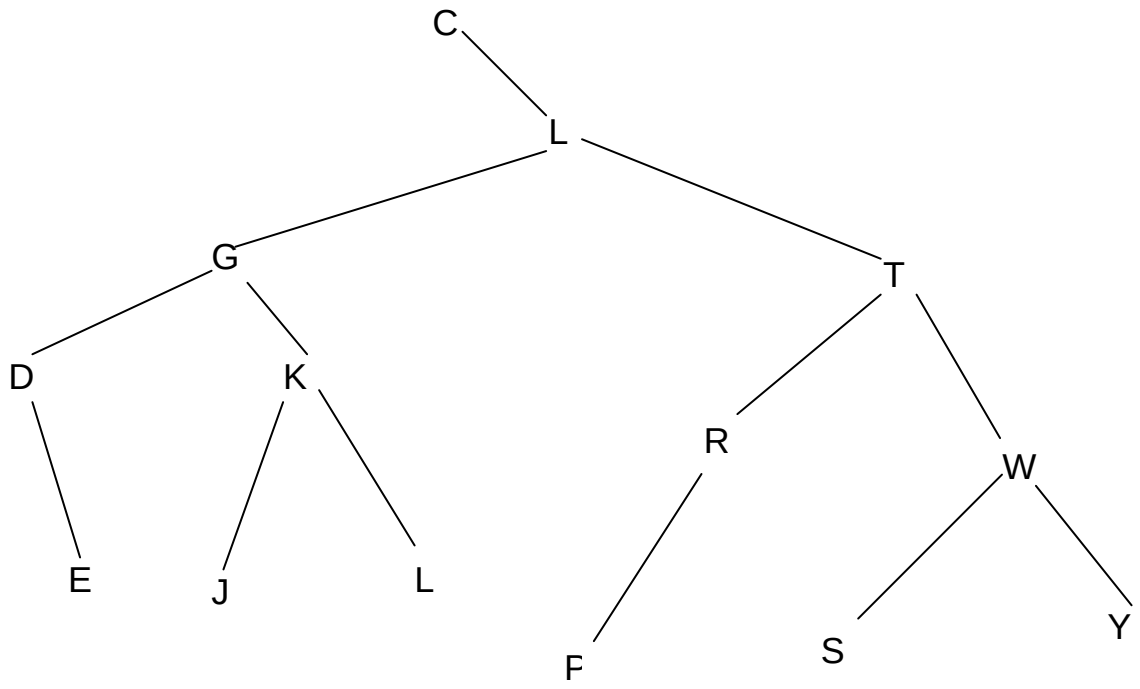
Consider the tree:



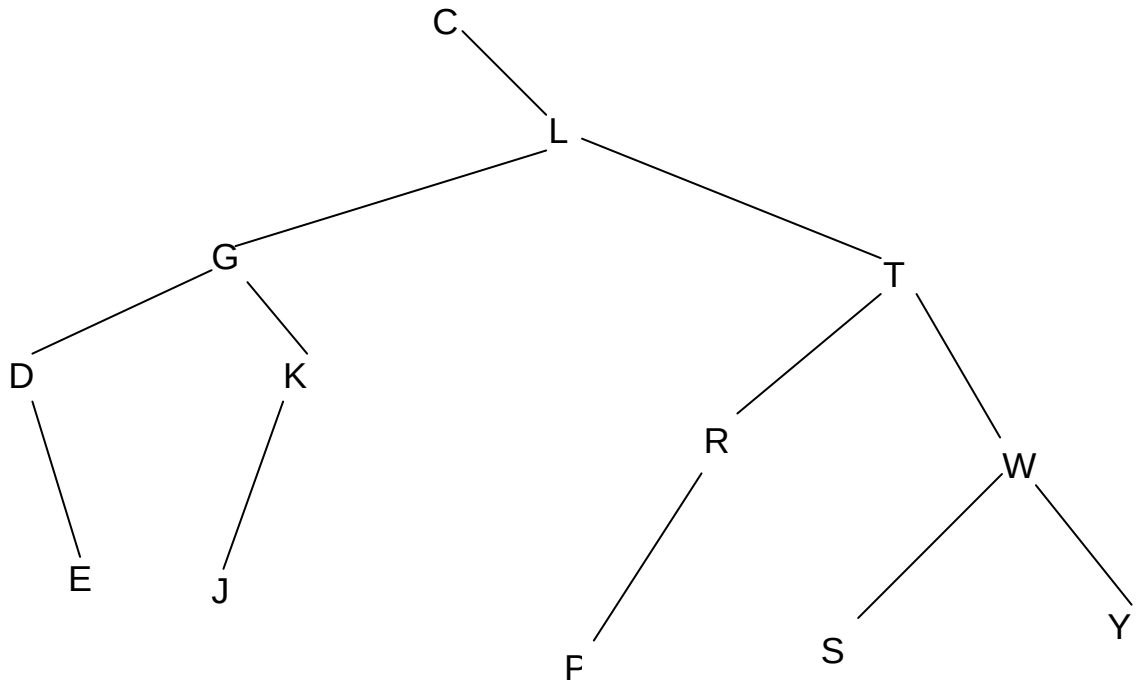
In order traversal: C D E G J K L **N** P R S T W Y
Now let us say we want to delete node N.
So let us find the rightmost node of the left subtree. In this tree this happens to be L.

All elements in the left subtree are going to be smaller than this node.

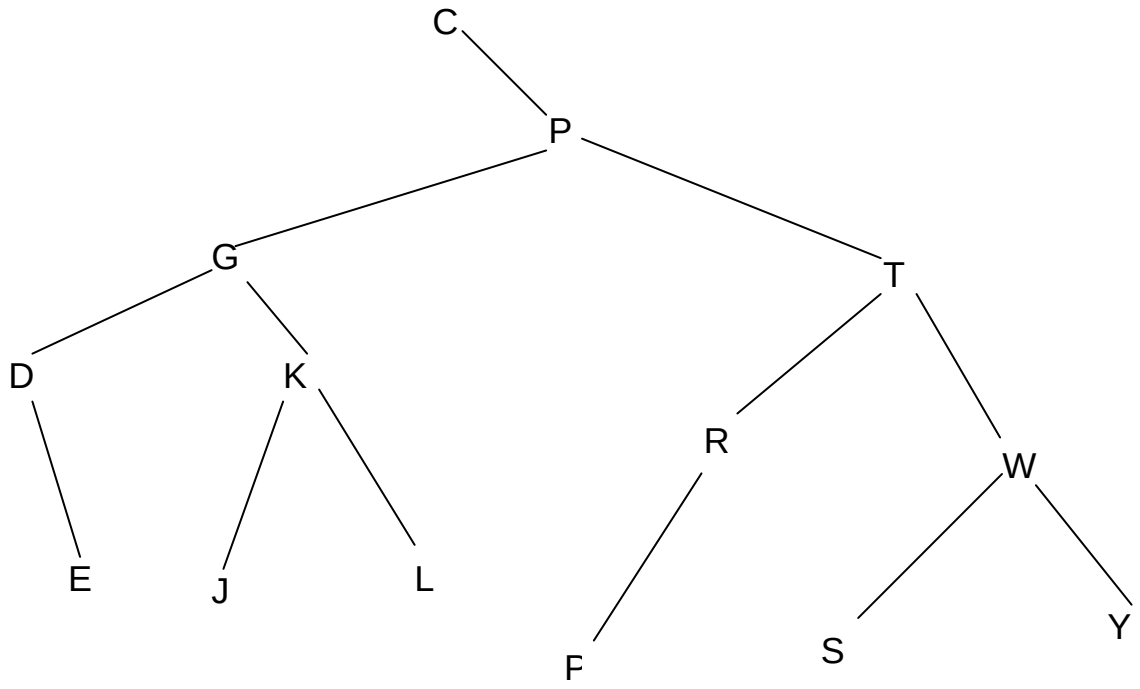
All elements in the right subtree are going to be greater than this node. It can simply replace N, while keeping the structure of the tree undisturbed. So copy the node L in N.



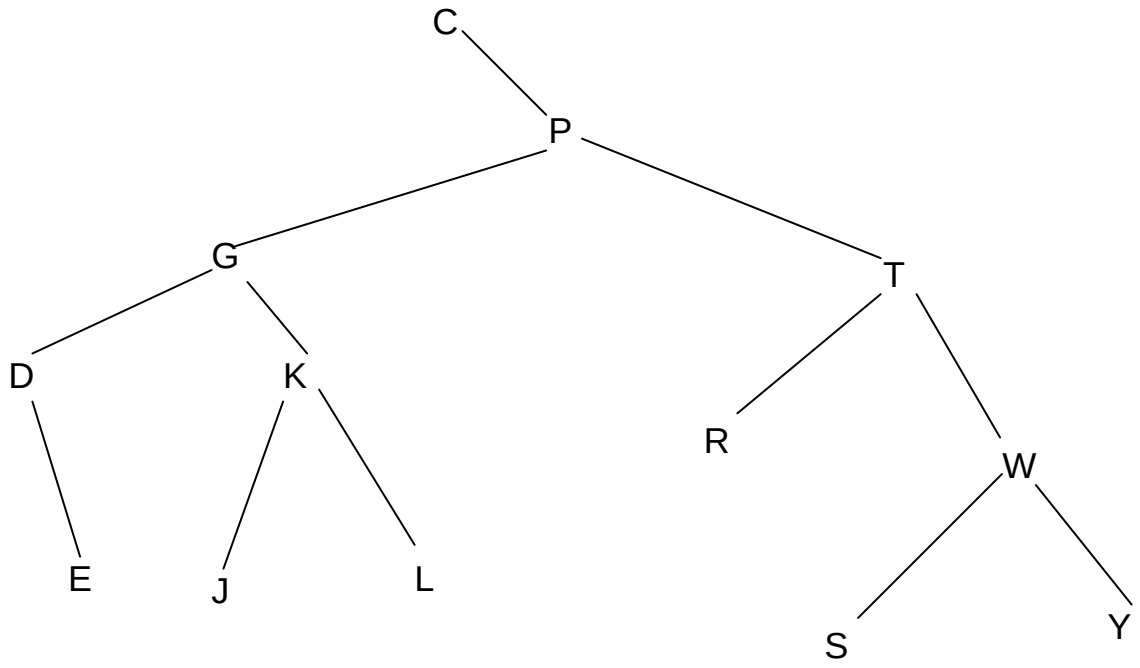
The value of L at the leaf node now can be deleted.



The node N can also be removed by replacing it with left most node of the right subtree and making the appropriate links. In this case node P becomes the right child of C



Now the leaf node P can be deleted ,resulting in the tree



The subtree G becomes the left child of P
And the subtree T becomes the right child of P.

In order traversal: C D E G J K L P R S T W Y

Exercise: In the tree drawn above, delete the node W.

[Hint: Replace it by the right most child of its left subtree. In this case there is just one element S. So simply replace W by S. Verify every time that the inorder traversal of the nodes turns out to be the alphabetical order.]

Practice Problems

Shown below are three problems for you to practice writing algorithms for operations on binary trees. Since the tree has a naturally occurring recursive definition, make your functions recursive.

1. Write a function that will count the number of leaf nodes in a binary tree.

2. Write a function that will find the height of a binary tree. The height of an empty tree is defined a zero. The height of a single node tree is defined as 1.

3. Write a function that will interchange all the left and right subtrees in a binary tree.