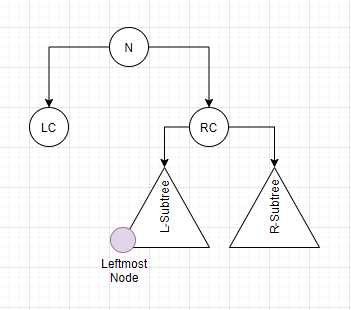
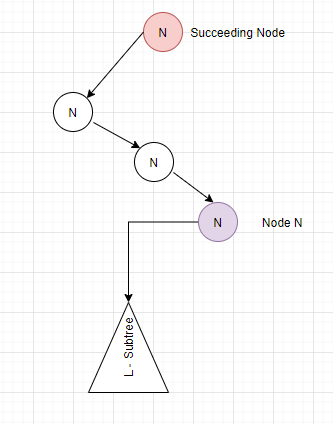
1. PostOrder Traversal is necessary if the action to be performed on the current node requires values from its children.  
   For example – calculating the max of a tree. We use PostOrderTraversal - recursively calculating the max of the left tree and max of the right tree and then comparing the max of both with the current node value.
2. PreOrder Traversal is used when we need to calculate paths from the root. Here as the path starts from the root and proceeds with a node from the next level of the tree, we use PreOrder Traversal.
3. The type of traversal becomes a moot point when the action to be performed does not depend on the values from its children. Here we simply do a recursion (ensuring the action that we need to perform is done for every node).
4. Recursion can be viewed from the following perspectives
   1. **Data Structure being broken down into smaller and similar data structures**.  
      For Example – A binary tree can be broken down into a left binary tree and right binary tree at its root. An instance of it is finding maximum of a tree. We say a maximum of a tree is nothing but the Max(root, Max(leftTree),Max(rightTree)).
   2. **An action to be repeatedly performed.**For example – Calculating the total sum of a binary tree, we simply perform recursion on the node’s children and keep adding the value to the sum(repeated action).
5. **The root of a tree represents the tree itself.**For example – FindMax(rootOfTree)  
    {  
    FindMax(leftchild);   
    FindMax(rightChild)  
    }   
    When we pass the children they **represent the root** of the left and right subtree.
6. **Preorder Traversal can be looked as a repetitive action that needs to be performed for every node.** In one call of the function we are performing the action on the current node and then calling the function recursively to perform the same action on its children.
7. **Postorder Traversal can be looked as solving a problem for the tree by solving the same problem for its left subtree and right subtree.** In one call of a function we first obtain the solution from the left subtree and right subtree and use the solutions for the current node.
8. **InorderTraversal can be viewed as moving from the leftmost node(the last node we reach when we keep following the left child from the root) to the rightmost node(the last node we reach when we keep following the right child from the root).  
   Leftmost Node = the first node in the inorder traversal ordering  
   Rightmost Node = the last node in the inorder traversal ordering  
     
   In the ordering the successor of a given node(N) is dependent on if N has a right subtree or not.**
   1. **Case 1 – It has a right subtree  
      Then the succeeding node is the leftmost node of it’s right subtree.**
   2. **Case 2 – There is no right subtree  
      In this case, the succeeding node is the first node for which move towards the right while going backwards (N->parent -> parent-> parent…..)**So when we add a node to the tree, obviously this node is a leafnode. The newly added node can be inserted anywhere in the *left-> right spectrum*. In the inorder traversal the newly added node will come to the *adjacent left of its parent if it is added as a left child* and will come to the *adjacent right of its parent if it is added as a right child*.