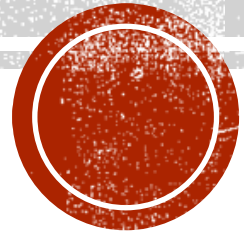
DATA STRUCTURES AND ALGORITHMS



INSERTION IN BST

- Check the value to be inserted (say **X**) with the value of the current node (say **val**) we are in:
  - If **X** is less than **val** move to the left subtree.
  - Otherwise, move to the right subtree.
- Once the leaf node is reached, insert **X** to its right or left based on the relation between **X** and the leaf node's value.

INSERTION IN BST

```
void BST::insert(int value)
{
        //Node create:
        Node*nn = new Node;
        nn->data = value;
        nn->leftChild = nullptr;
        nn->rightChild = nullptr;

        if (root == nullptr) //empty tree case
                root = nn;

        else { -------------- }
```

TREE TRAVERSAL

- Traversal is a process to visit all the nodes of a tree and may print their values too.
- Because, all nodes are connected via edges (links) we always start from the root (head) node. That is, we cannot randomly access a node in a tree.
- There are three ways which we use to traverse a tree –
  - In-order Traversal
  - Pre-order Traversal
  - Post-order Traversal

IN-ORDER TRAVERSAL

- In this traversal method, traversal method is as follows:
  - the left subtree is visited first,
  - then the root
  - then the right sub-tree.
- We should always remember that every node may represent a subtree itself.
- If a binary tree is traversed **in-order**, the output will produce sorted key values in an ascending order.

PRE-ORDER TRAVERSAL

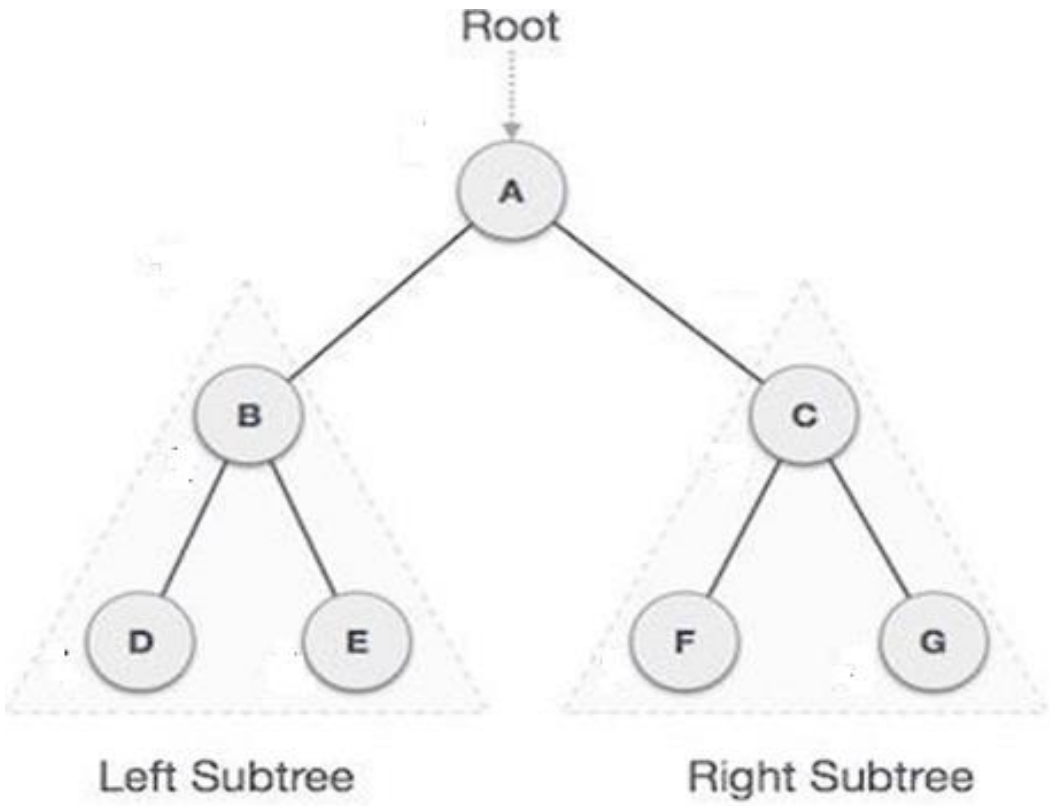
The steps to perform the preorder traversal are listed as follows -

- First, visit the root node.
- Then, visit the left subtree.
- At last, visit the right subtree.

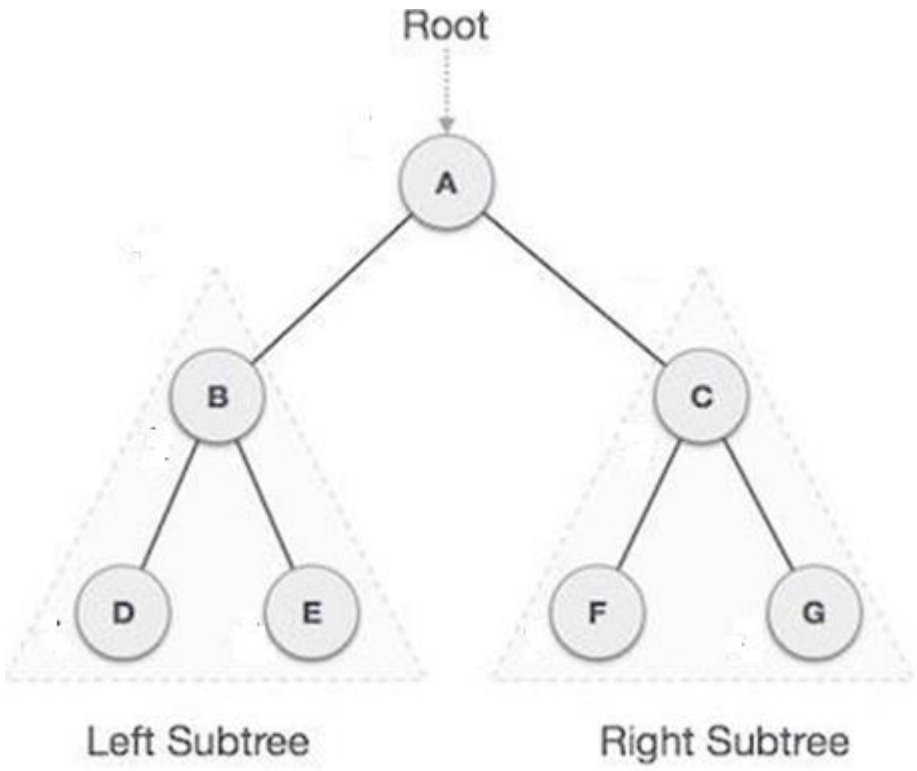
POST-ORDER TRAVERSAL

- Post-order traversal is used to get the postfix expression of a tree.
- The following steps are used to perform the post-order traversal:
- Traverse the left subtree
- Traverse the right subtree
- Root node is traversed at last

TREE TRAVERSAL

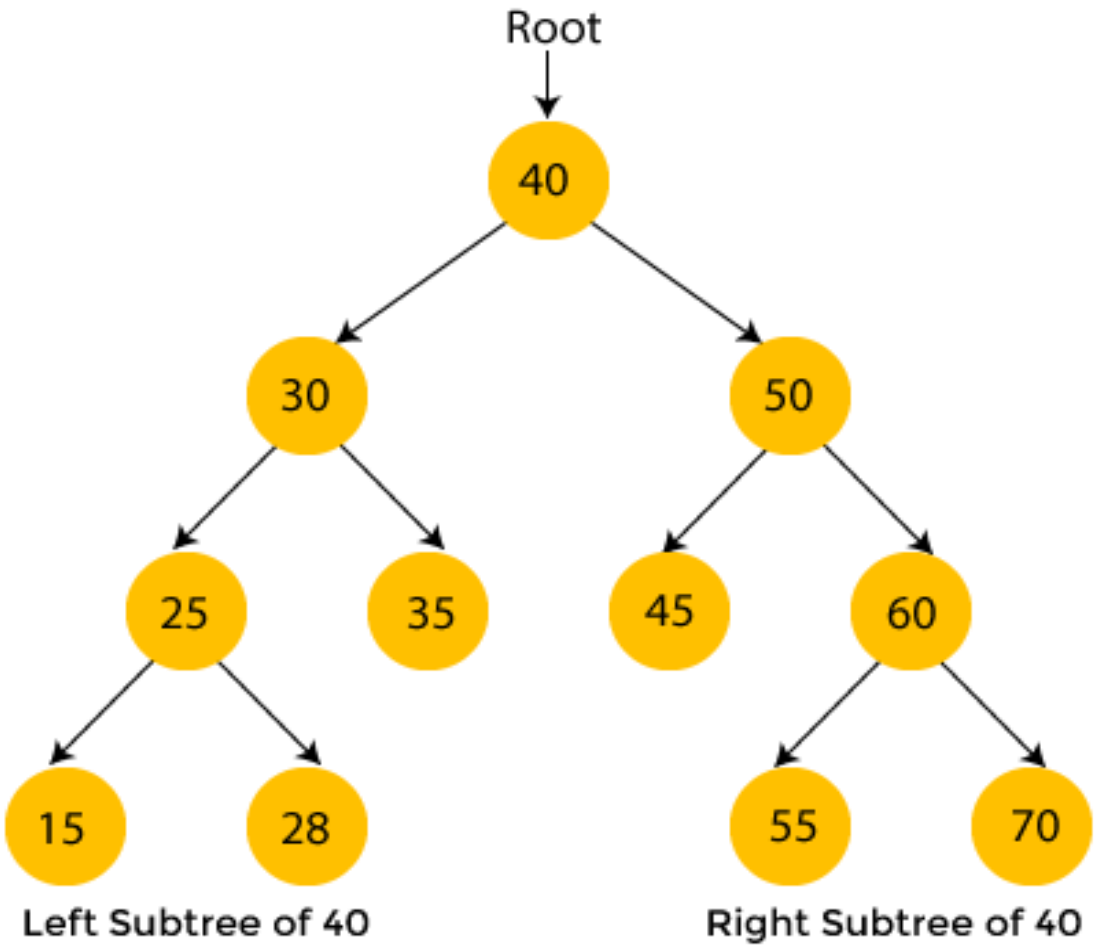


TREE TRAVERSAL

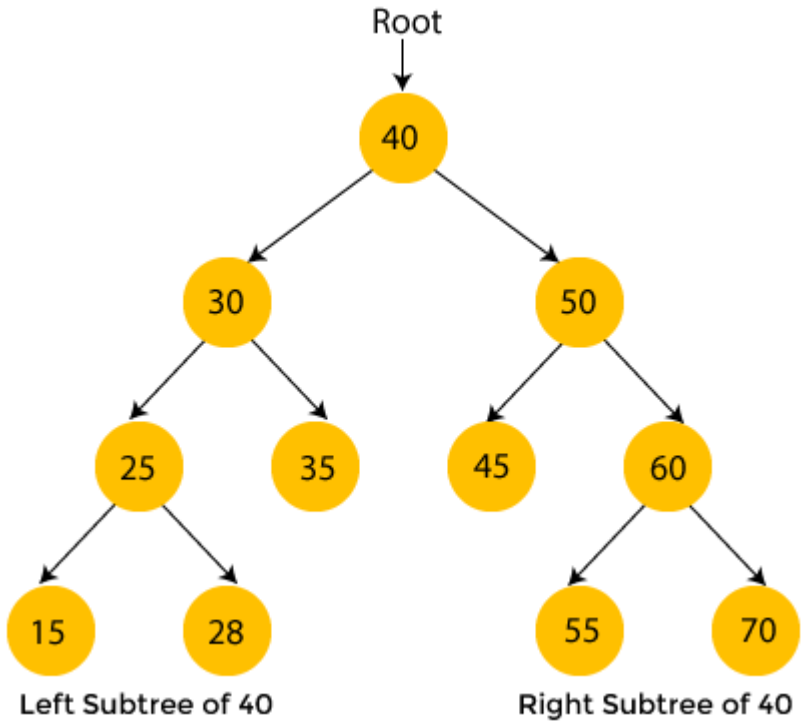


- Post-order: D → E → B → F → G → C → A
- Pre-order: A → B → D → E → C → F → G
- In-order: D → B → E → A → F → C → G

TREE TRAVERSAL



TREE TRAVERSAL

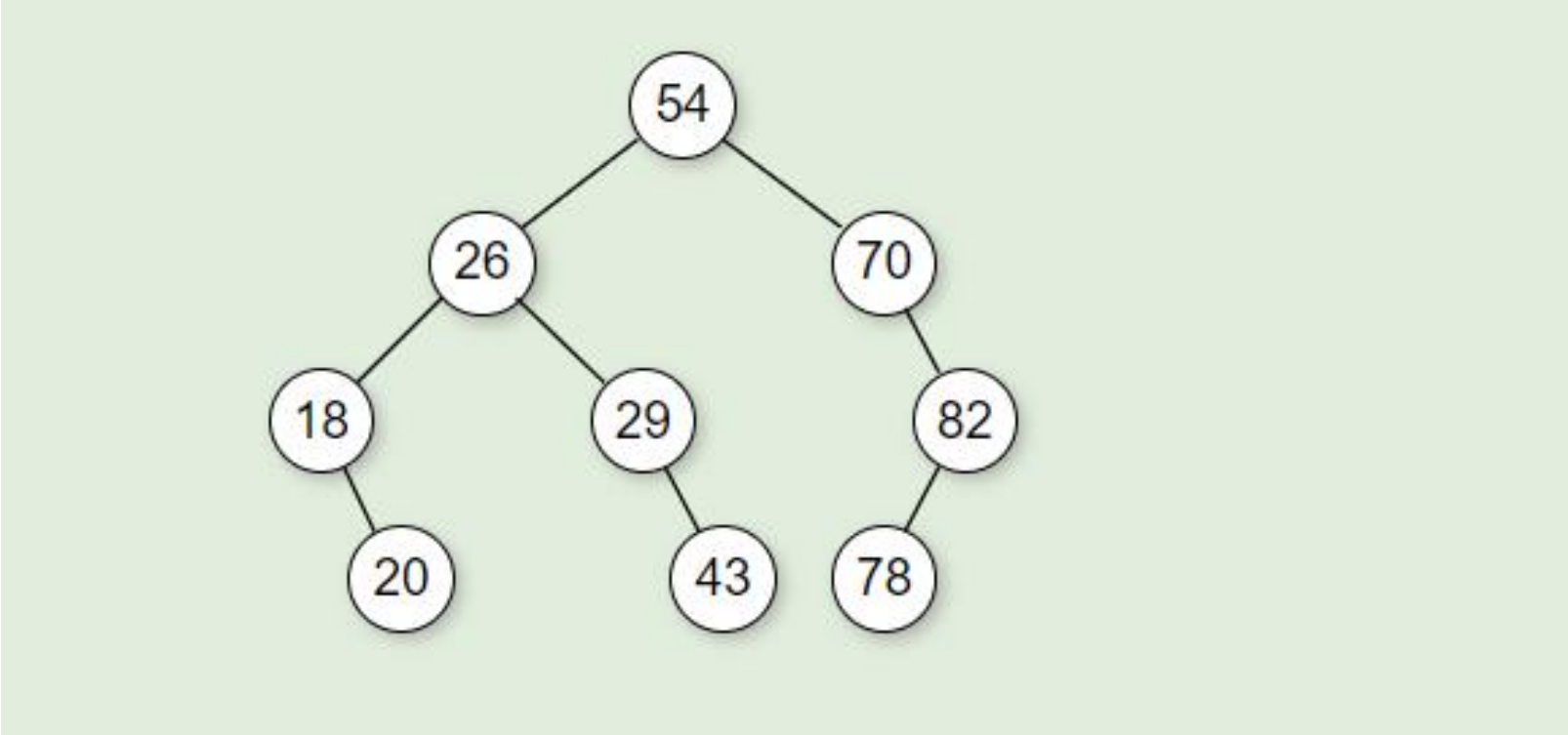


- **Pre-order:** 40, 30, 25, 15, 28, 35, 50, 45, 60, 55, 70
- Post-order: 15, 28, 25, 35, 30, 45, 55, 70, 60, 50, 40
- In-order: 15, 25, 28, 30, 35, 40, 45, 50, 55, 60, 70

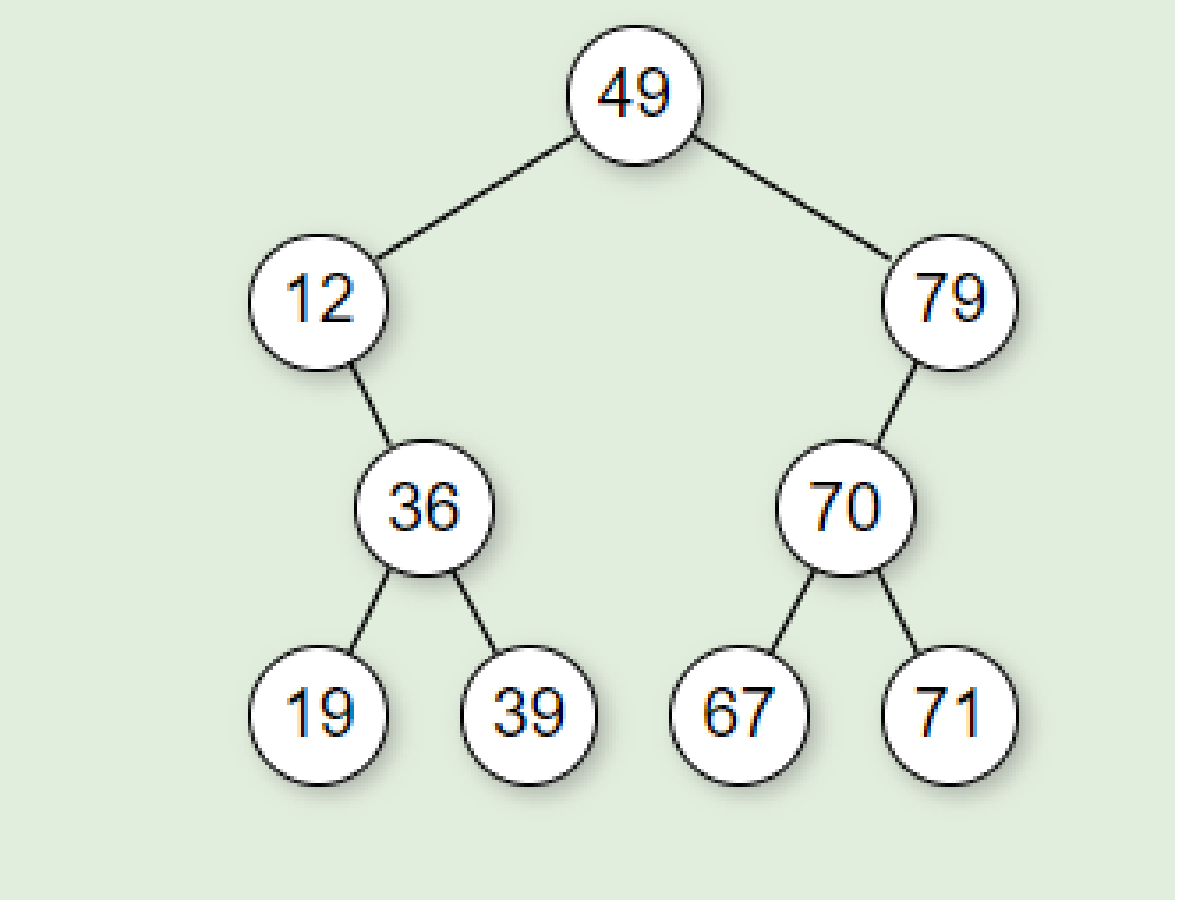
IN-ORDER TRVERSAL

```
void BST::INORDER(Node*p)
{
	if (p != nullptr)
	{
		INORDER(p->leftChild);
		cout << p->data << endl;
		INORDER(p->rightChild);
	}
}
```

TREE TRAVERSAL



TREE TRAVERSAL

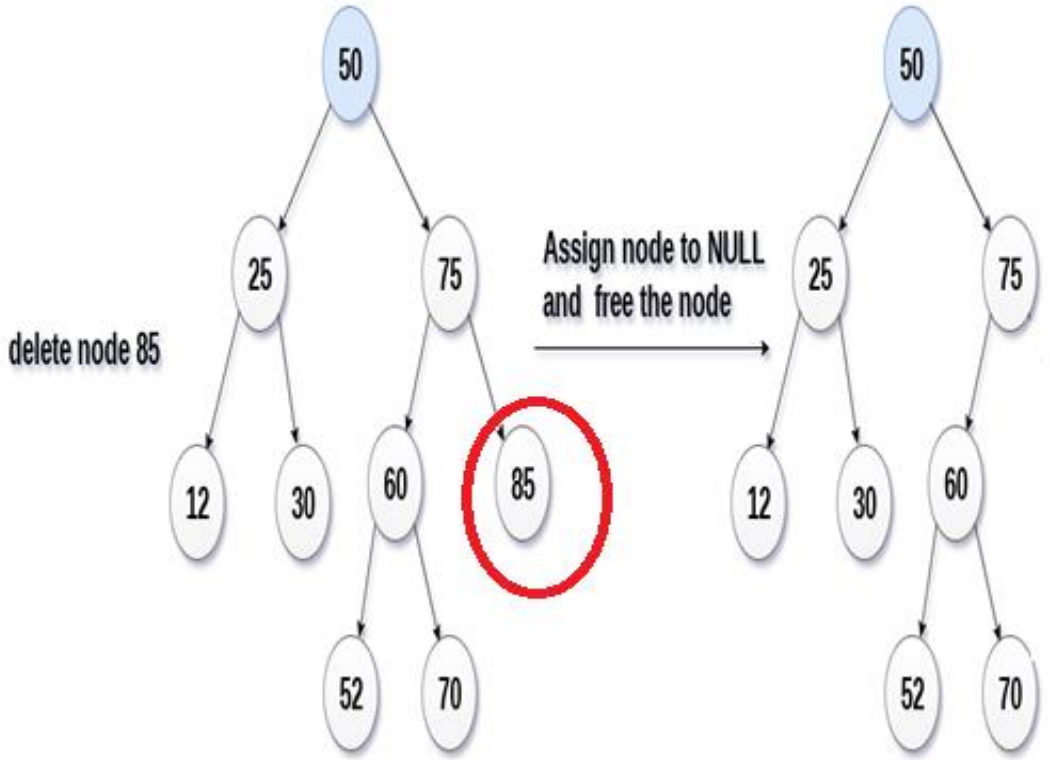


DELETION IN BST

- Delete function is used to delete the specified node from a binary search tree.
- However, we must delete a node from a binary search tree in such a way, that the property of binary search tree doesn't violate.
- There are three situations of deleting a node from binary search tree.
  - The node to be deleted is a leaf node
  - The node to be deleted has only one child
  - The node to be deleted has two children.

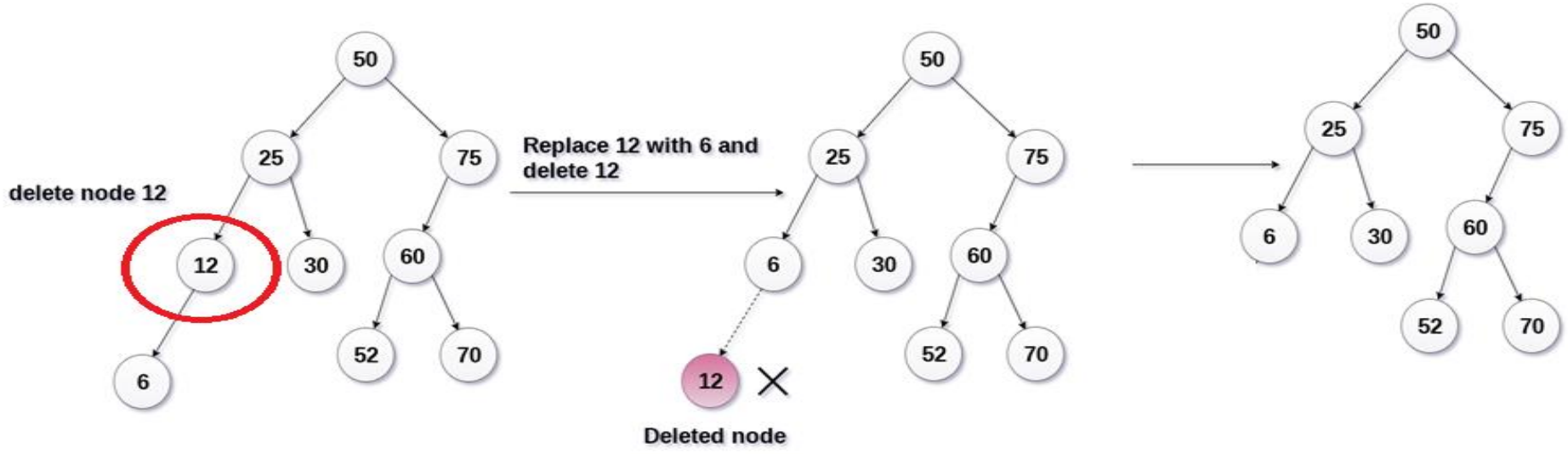
NODE TO BE DELETED IS A LEAF NODE

- It is the simplest case, in this case, replace the leaf node with the NULL and simple free the allocated space.
- In the following image, we are deleting the node 85, since the node is a leaf node, therefore the node will be replaced with NULL and allocated space will be freed.



NODE TO BE DELETED HAS ONE CHILD

- In this case, replace the node with its child and delete the child node, which now contains the value which is to be deleted. Simply replace it with the NULL and free the allocated space.
- In the following image, the node 12 is to be deleted. It has only one child. The node will be replaced with its child node and the replaced node 12 (which is now leaf node) will simply be deleted.



NODE TO BE DELETED HAS TWO CHILDREN

- It is a bit complexed case compare to other two cases.
- However, the node which is to be deleted, is replaced with its in-order successor or predecessor i.e. find the smallest node in the right subtree.
- Replace the value of current node with the in-order successor or predecessor
- After the procedure, replace the node with NULL and free the allocated space.

NODE TO BE DELETED HAS TWO CHILDREN