SNS COLLEGE OF ENGINEERING

Kurumbapalayam(Po), Coimbatore – 641 107
Accredited by NAAC-UGC with ‘A’ Grade
Approved by AICTE, Recognized by UGC & Affiliated to Anna University, Chennai

Department of Artificial Intelligence and Data Science

Course Name: 23ITB201 Data structures and Algorithms

II Year / III semester

Unit IV –Tree ADT

Topic: Binary search Tree

Binary Search Tree

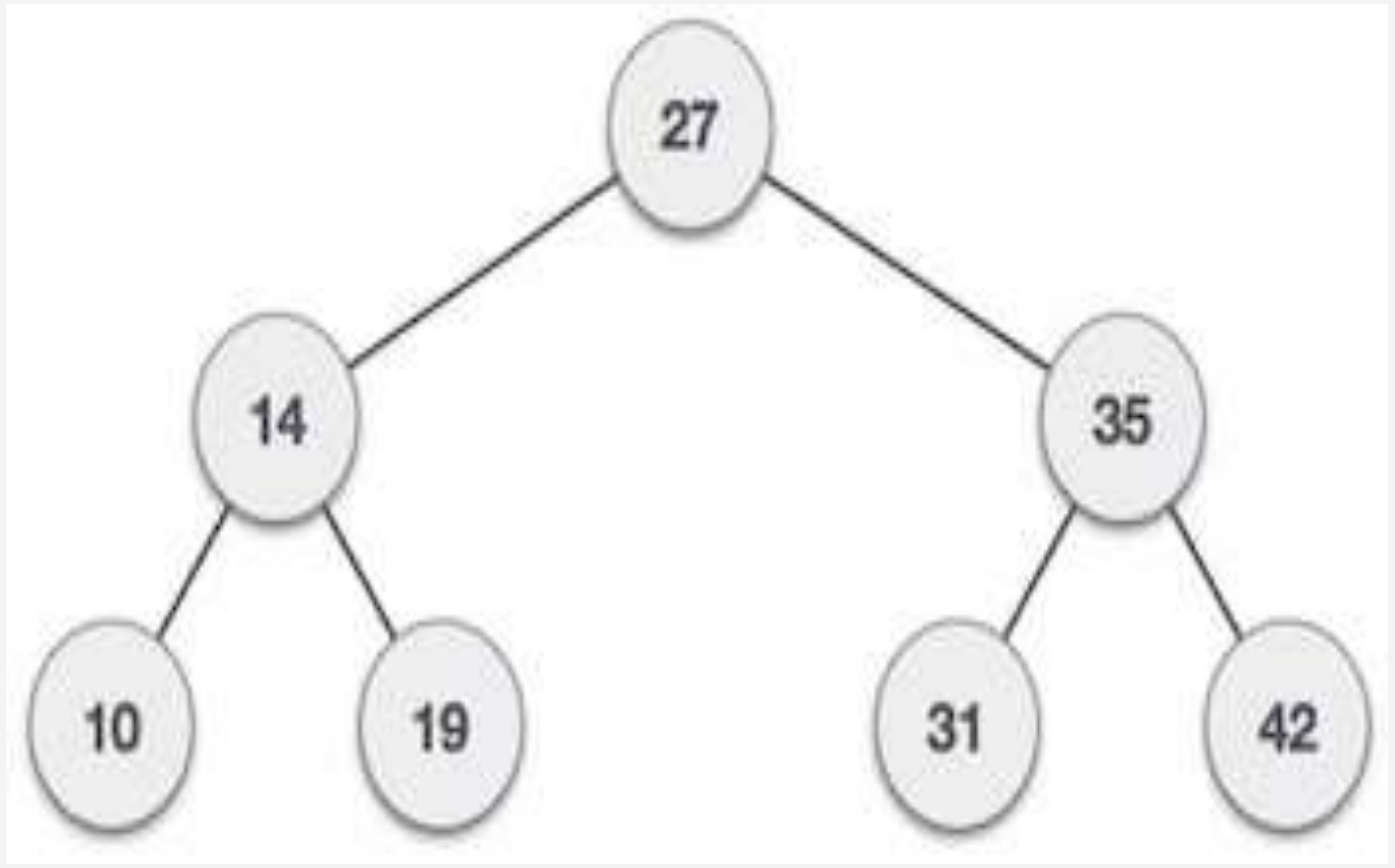
- Binary Search Tree is a node-based binary tree data structure which has the following properties:
- The left subtree of a node contains only nodes with keys lesser than the node’s key.
- The right subtree of a node contains only nodes with keys greater than or equal to the node’s key.
- The left and right subtree each must also be a binary search tree.

  left_subtree (keys) < node (key) ≤ right_subtree (keys)
- Binary Search Tree is used to maintain sorted stream of data.

Binary Search Tree

Representation

- BST is a collection of nodes arranged in a way where they maintain BST properties.



We observe that the root node key (27) has all less-valued keys on the left sub-tree and the higher valued keys on the right sub-tree.

Binary Search Tree

Operations on a Binary Search Tree

The following operations are performed on a binary search tree.

1. Search

2. Insertion

3. Deletion

Binary Search Tree

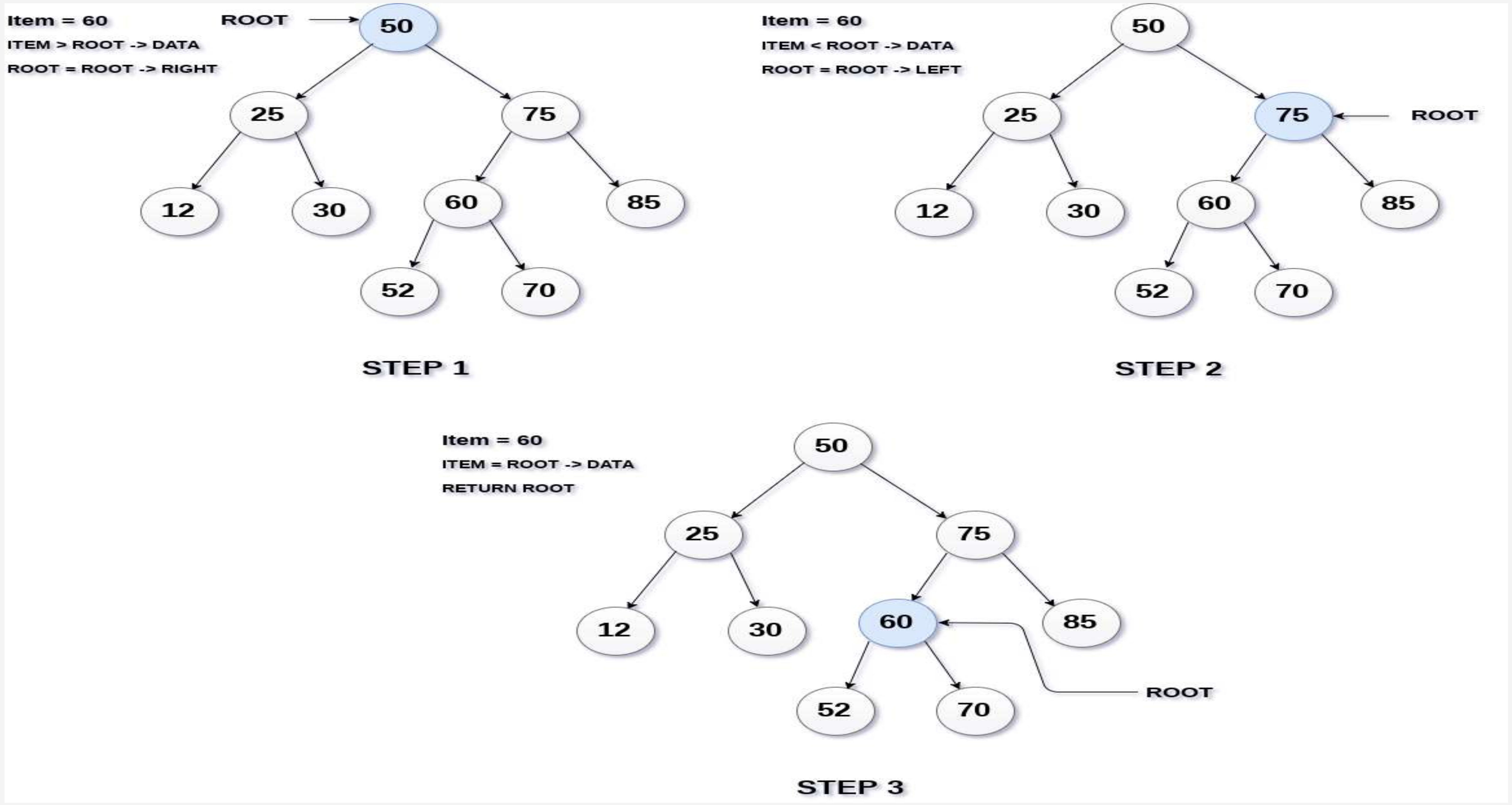
Search operation in BST

Steps involved in Search operation:

- Compare the element with the root of the tree.
- If the item is matched then return the location of the node.
- Otherwise check if item is less than the element present on root, if so then move to the left sub-tree.
- If not, then move to the right sub-tree.
- Repeat this procedure recursively until match found.
- If element is not found then return NULL.

Binary Search Tree

Search Item = 60



Binary Search Tree

Structure Definition of Binary Tree

```
struct node{
   int key;
   struct node *left, *right;
};
```

Search Routine

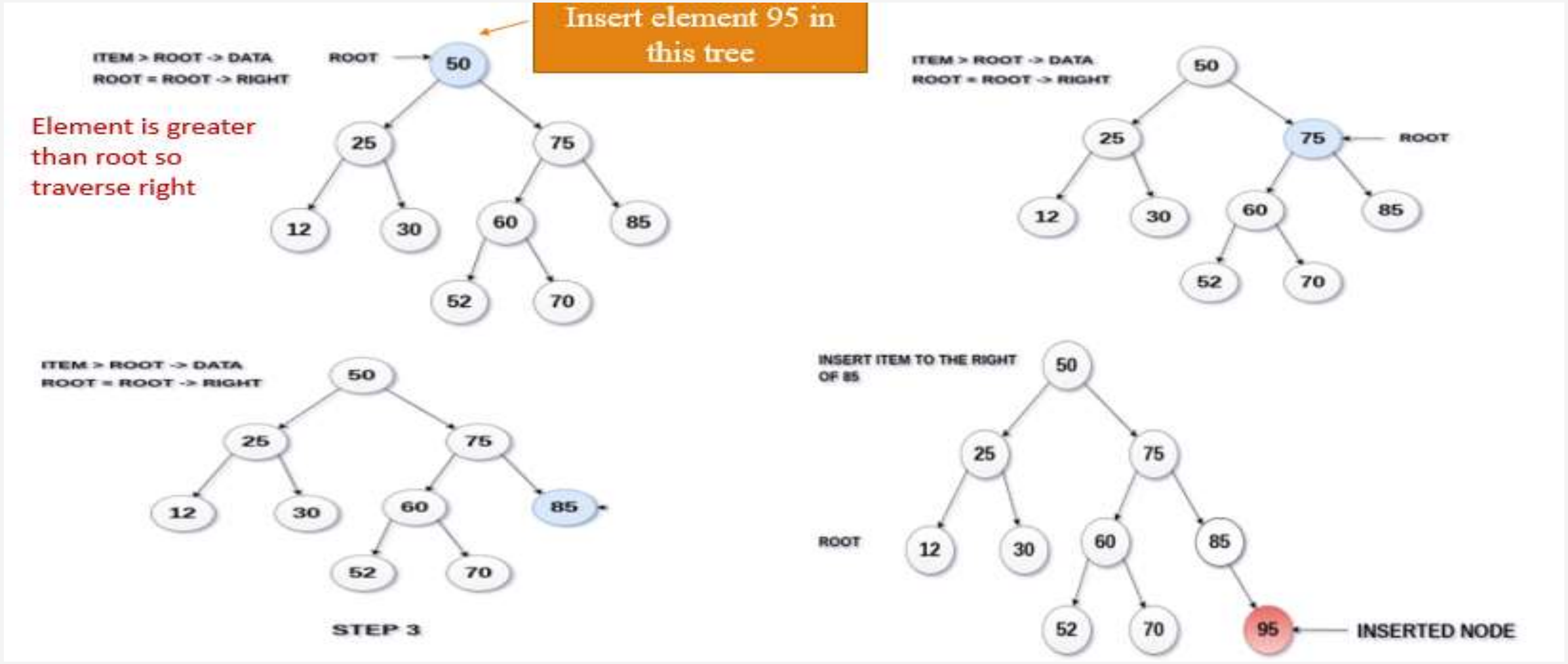
```
struct node* search(struct node* root, int key){
   if (root == NULL || root->key == key)
      return root;
   if (root->key < key)
      return search(root->right, key);
   return search(root->left, key);
}
```

Insertion Operation in BST

Insertion Operation in BST

1. Allocate the memory for tree node.
2. Set the data part to the value and set the left and right pointer of tree, point to NULL.
3. If the item to be inserted, will be the first element of the tree, then the left and right of this node will point to NULL.
4. Else, check if the item is less than the root element of the tree, if this is true, then recursively perform this operation with the left of the root.
5. If this is false, then perform this operation recursively with the right sub-tree of the root.

Insertion Operation in BST



Insertion Operation in BST

Insertion Routine

```
struct node* insert(struct node* node, int key){
   if (node == NULL) return newNode(key);
      if (key < node->key)
         node->left = insert(node->left, key);
      else if (key > node->key)
         node->right = insert(node->right, key);
   return node;
}
```

Deletion operation in BST

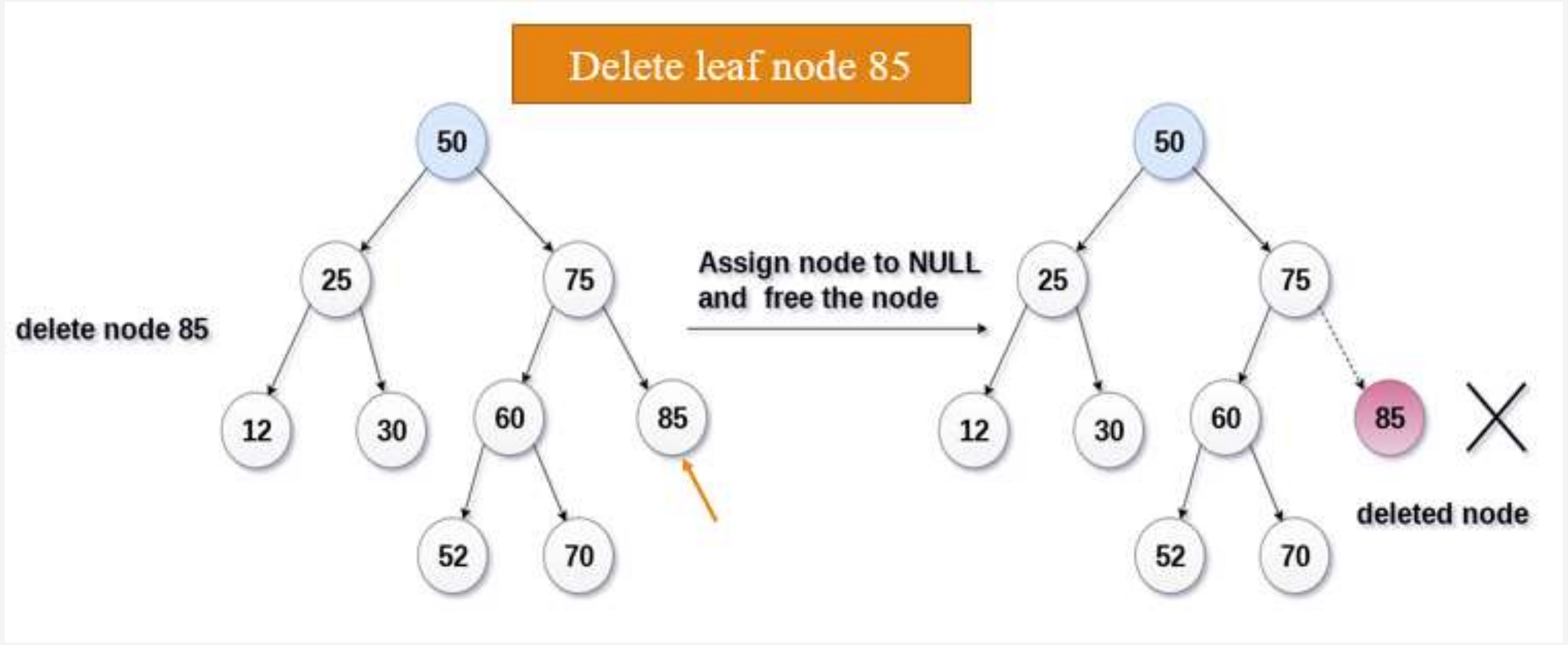
Deletion operation in BST:

- 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.

Deletion operation in BST

The node to be deleted is a leaf node

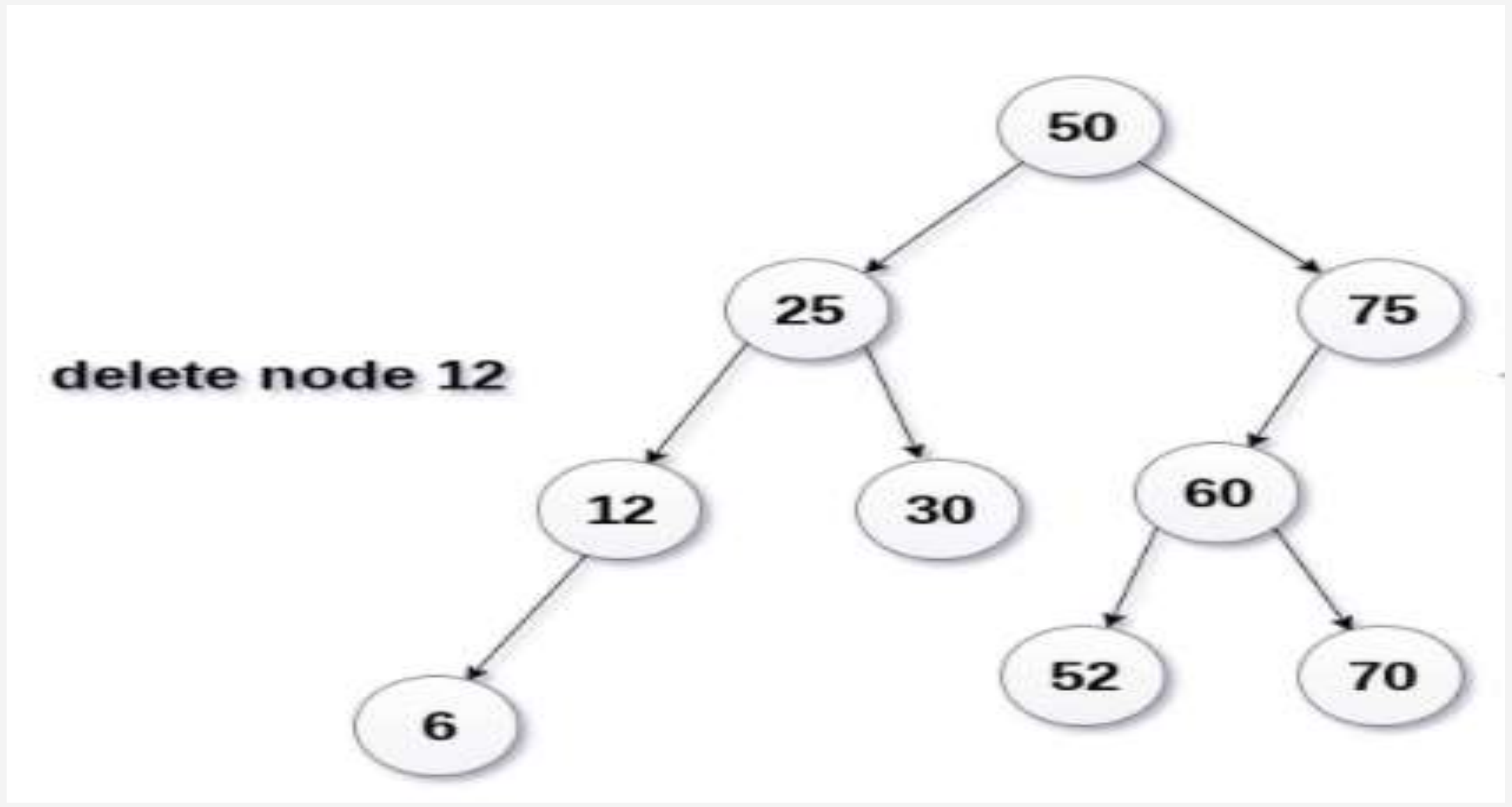
- It is the simplest case, in this case, replace the leaf node with the NULL and simple free the allocated space.



Deletion operation in BST

The node to be deleted has only one child.

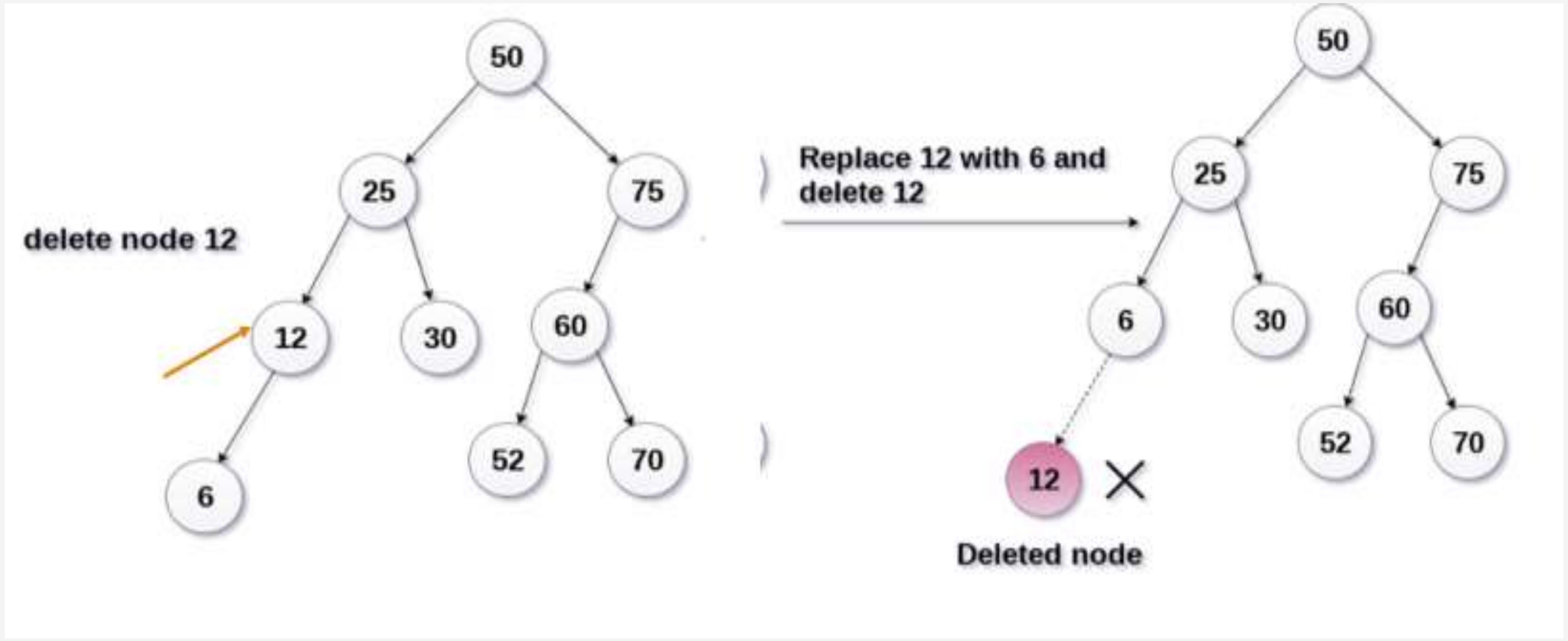
- In this case, replace the node with its child and delete the child node, which now contains the value which is to be deleted.
- Simply replace it with the NULL and free the allocated space.



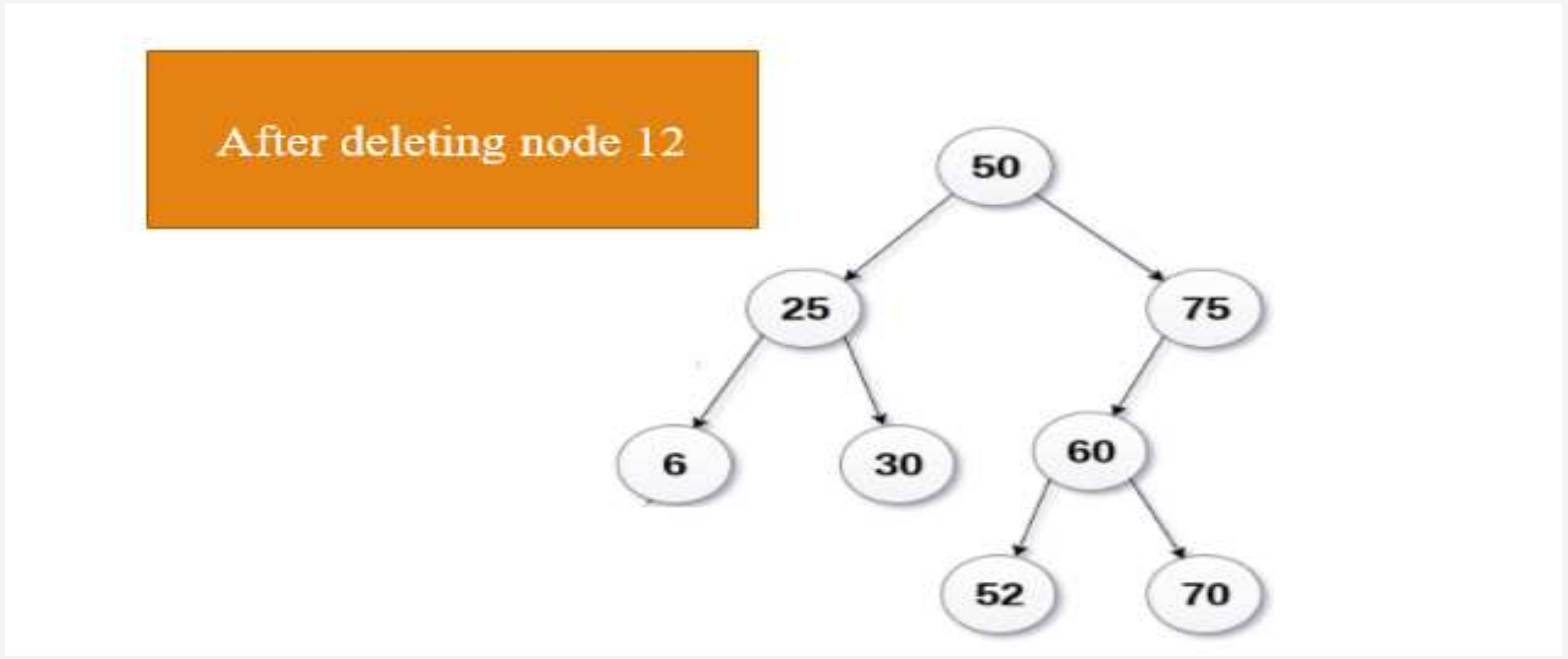
Deletion operation in BST

The node to be deleted has only one child:

- 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.



Deletion operation in BST



Deletion operation in BST

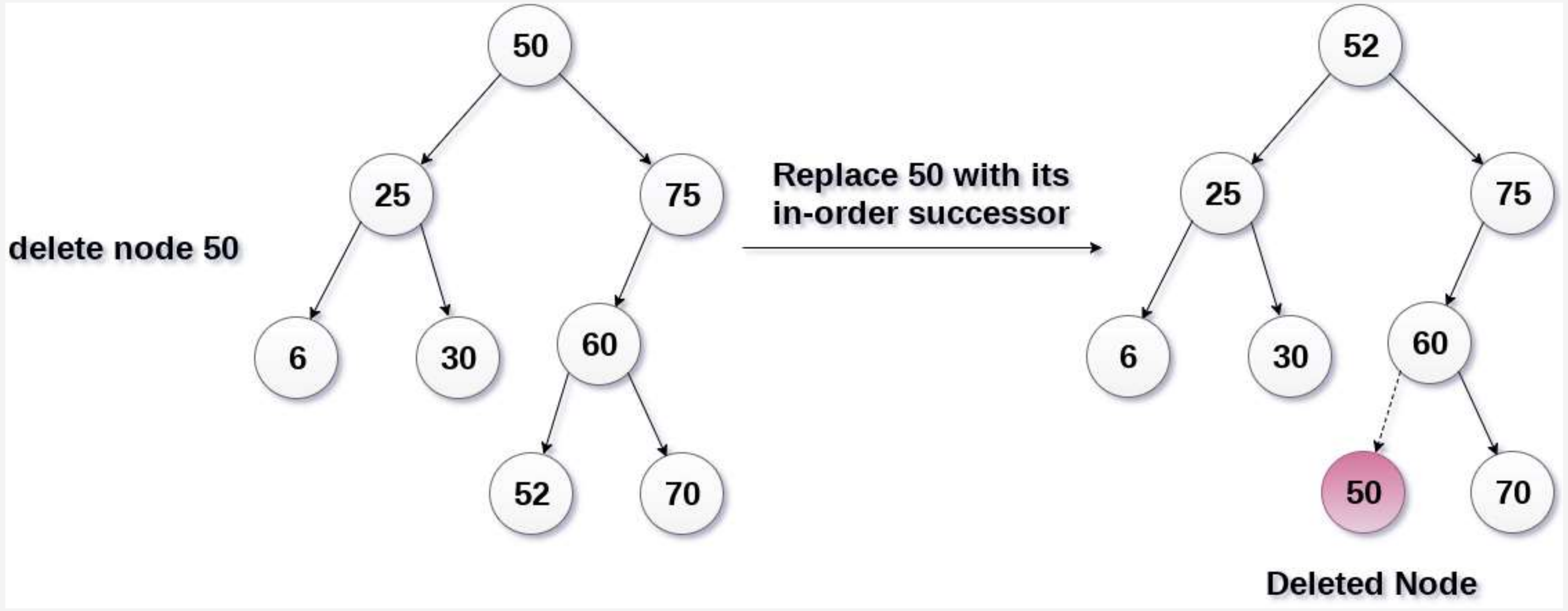
The node to be deleted has two children:

- The node which is to be deleted, is replaced with its in-order successor or predecessor recursively until the node value (to be deleted) is placed on the leaf of the tree.
- After the procedure, replace the node with NULL and free the allocated space.

The node to be deleted has two children.

In the following image, the node 50 is to be deleted which is the root node of the tree

Deletion operation in BST



The in-order traversal of the tree 6, 25, 30, 50, 52, 60, 70, 75.

Replace 50 with its in-order successor 52.

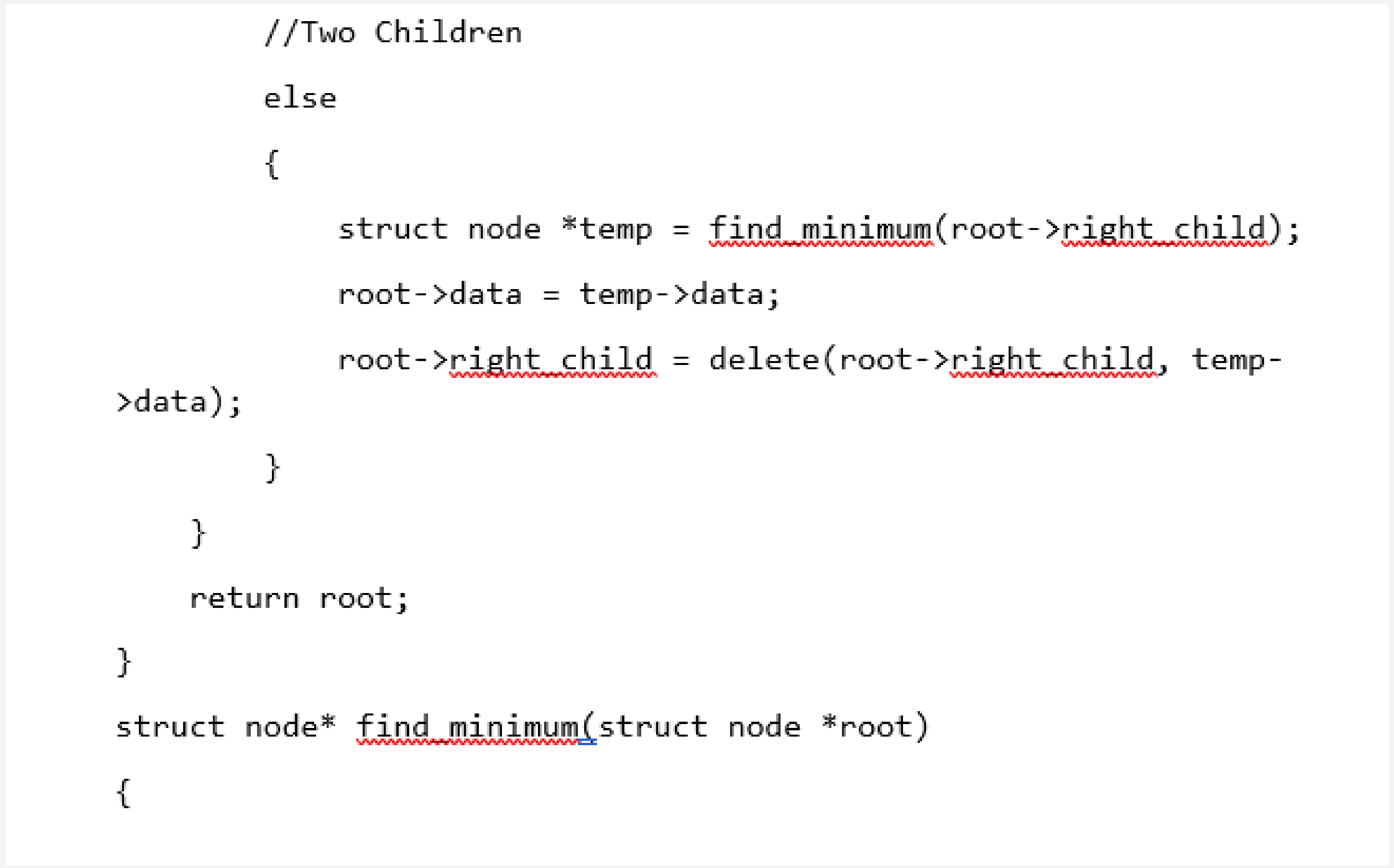
Deletion operation in BST

Delete Routine

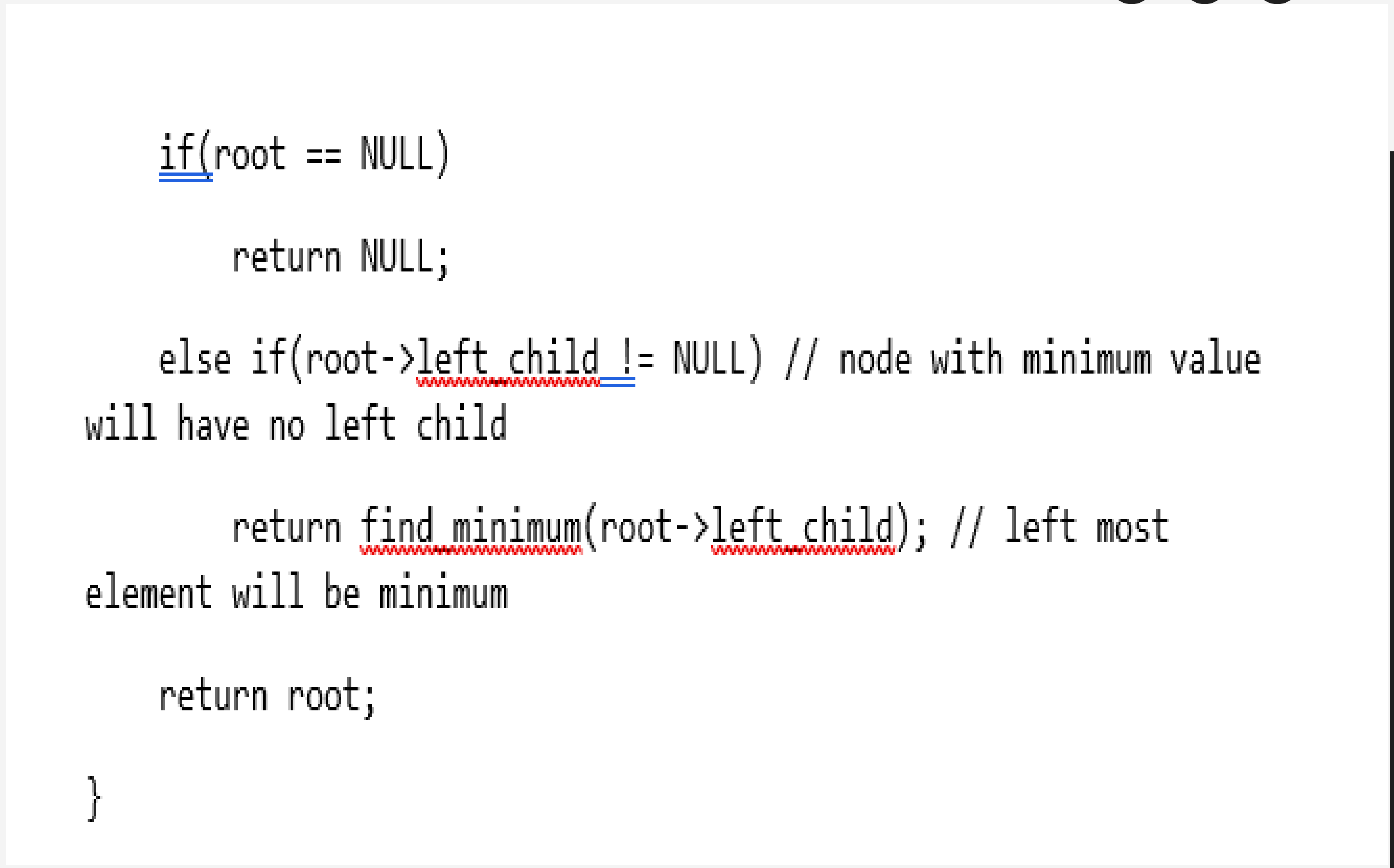
```
struct node* delete(struct node *root, int x)
{
    //searching for the item to be deleted
    if(root==NULL)
        return NULL;
    if (x>root->data)
        root->right_child = delete(root->right_child, x);
    else if(x<root->data)
        root->left_child = delete(root->left_child, x);
    else
    {
```

```
        //No Children
        if(root->left_child==NULL && root->right_child==NULL)
        {
            free(root);
            return NULL;
        }
    //One Child
        else if(root->left_child==NULL || root-
    >right_child==NULL)
        {
            struct node *temp;
            if(root->left_child==NULL)
                temp = root->right_child;
            else
                temp = root->left_child;
            free(root);
            return temp;
        }
```

Deletion operation in BST

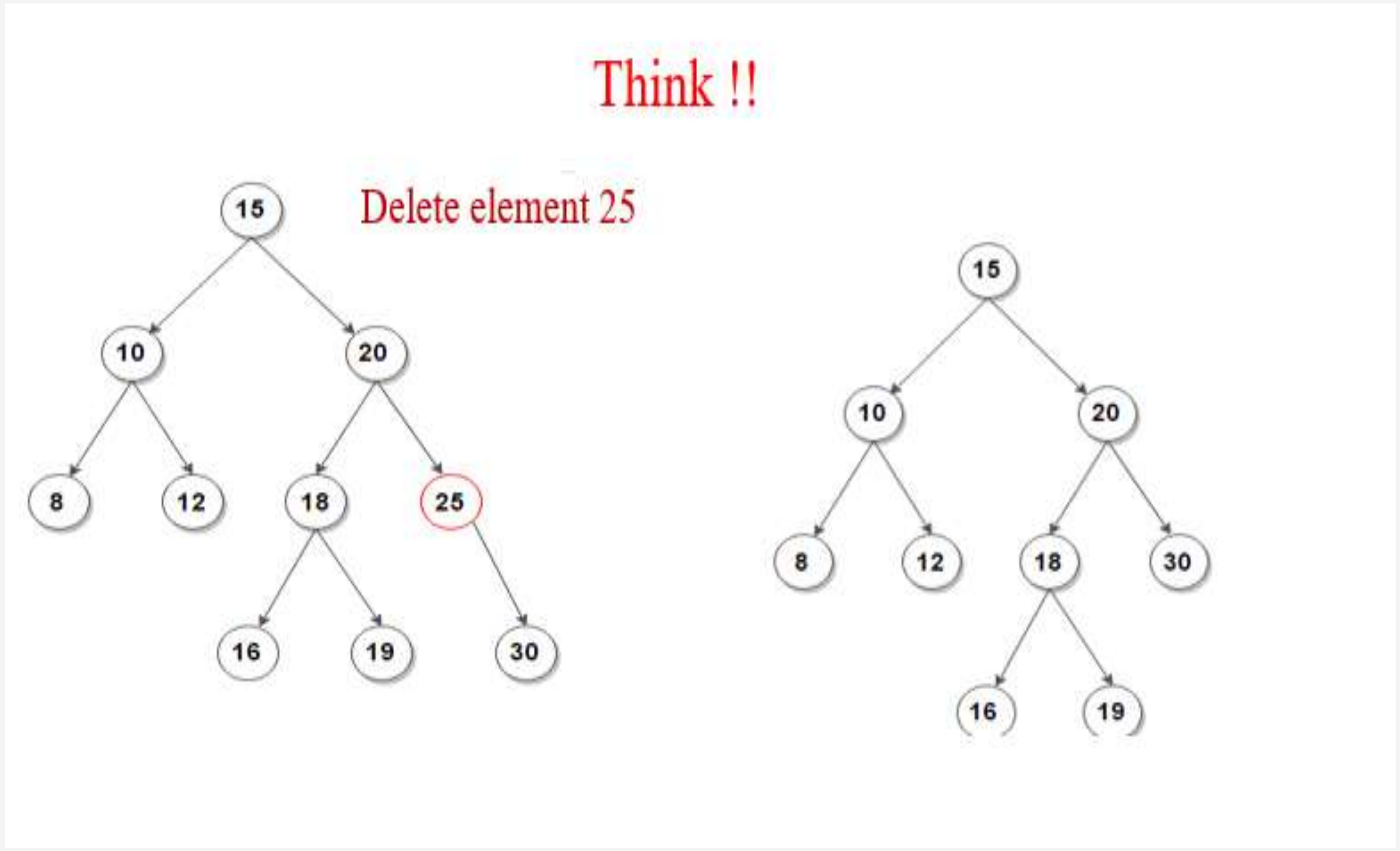


```
            //Two Children
            else
            {
                struct node *temp = find_minimum(root->right_child);
                root->data = temp->data;
                root->right_child = delete(root->right_child, temp-
>data);
            }
        }
        return root;
    }
    struct node* find_minimum(struct node *root)
    {
```



```
        if(root == NULL)
            return NULL;
        else if(root->left_child != NULL) // node with minimum value
    will have no left child
            return find_minimum(root->left_child); // left most
    element will be minimum
        return root;
    }
```

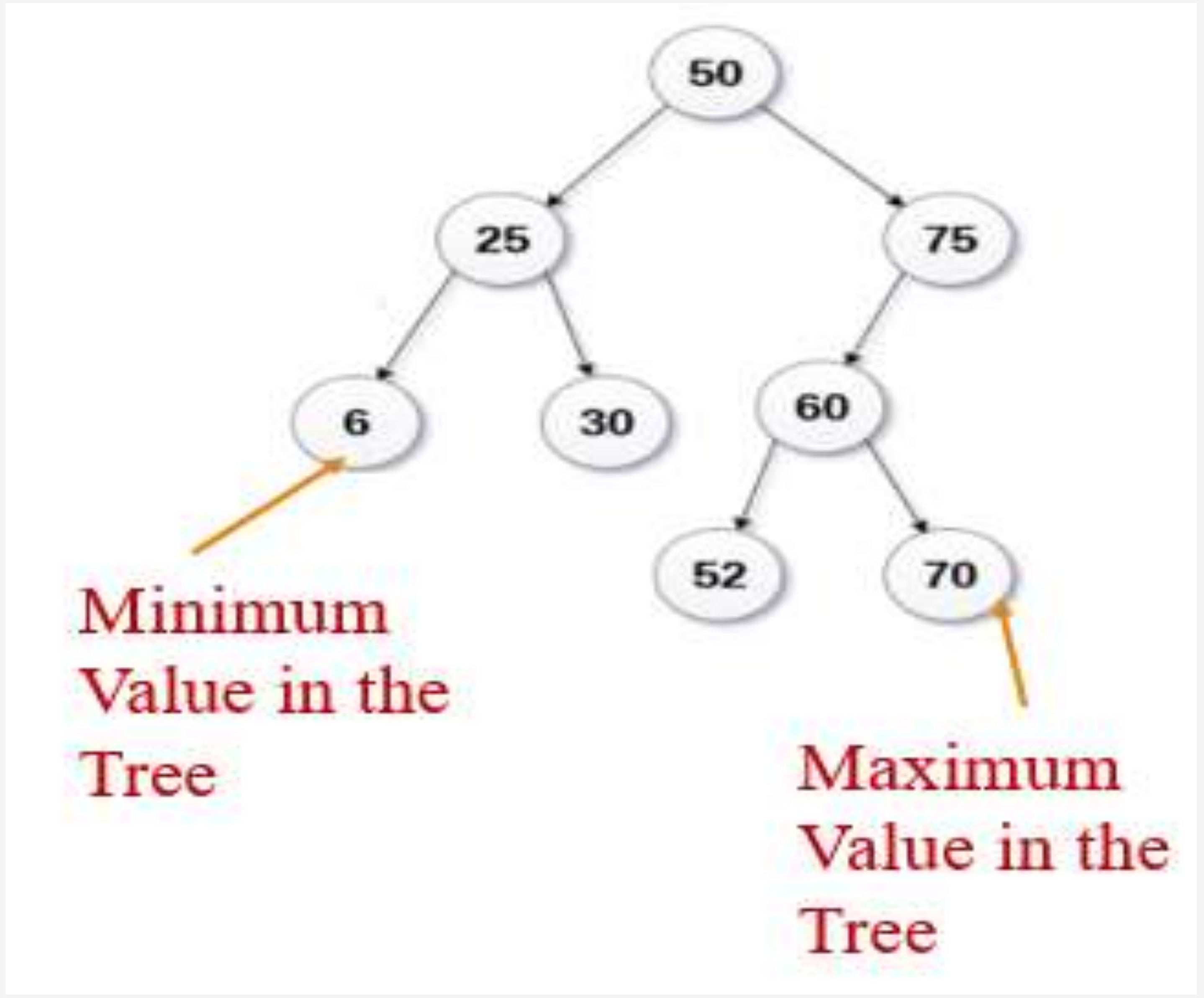
Deletion operation in BST



Finding Minimum and Maximum Value in BST

Finding Minimum and Maximum Value in BST

```
void minimum(struct node *root)
{
    while(root != NULL && root->left != NULL)
    {
        root = root->left;
    }
    printf("\nSmallest value is %d\n", root->info);
}
void maximum(struct node *root)
{
    while (root != NULL && root->right != NULL)
    {
        root = root->right;
    }
    printf("\nLargest value is %d", root->info);
}
```


50
25
75
6
30
60
52
70
Minimum Value in the Tree
Maximum Value in the Tree