Algorithms

Lecture 3

Tree Data Structure (2)

A. S. M. Sanwar Hosen

Email: sanwar@wsu.ac.kr

Date: 23 March, 2023

Tree Traversal

- **Definition:** Traversal is a process to visit all the nodes of a tree and may access their values. The types of the traversals are:

- ✓ **Depth-First Traversal/Search (DFS):** A traversal or searching algorithm in data structures that explores all the nodes by going forward if possible or uses backtrack. There are three different types of DFS:

  1. Inorder Traversal
  2. Preorder Traversal
  3. Postorder Traversal

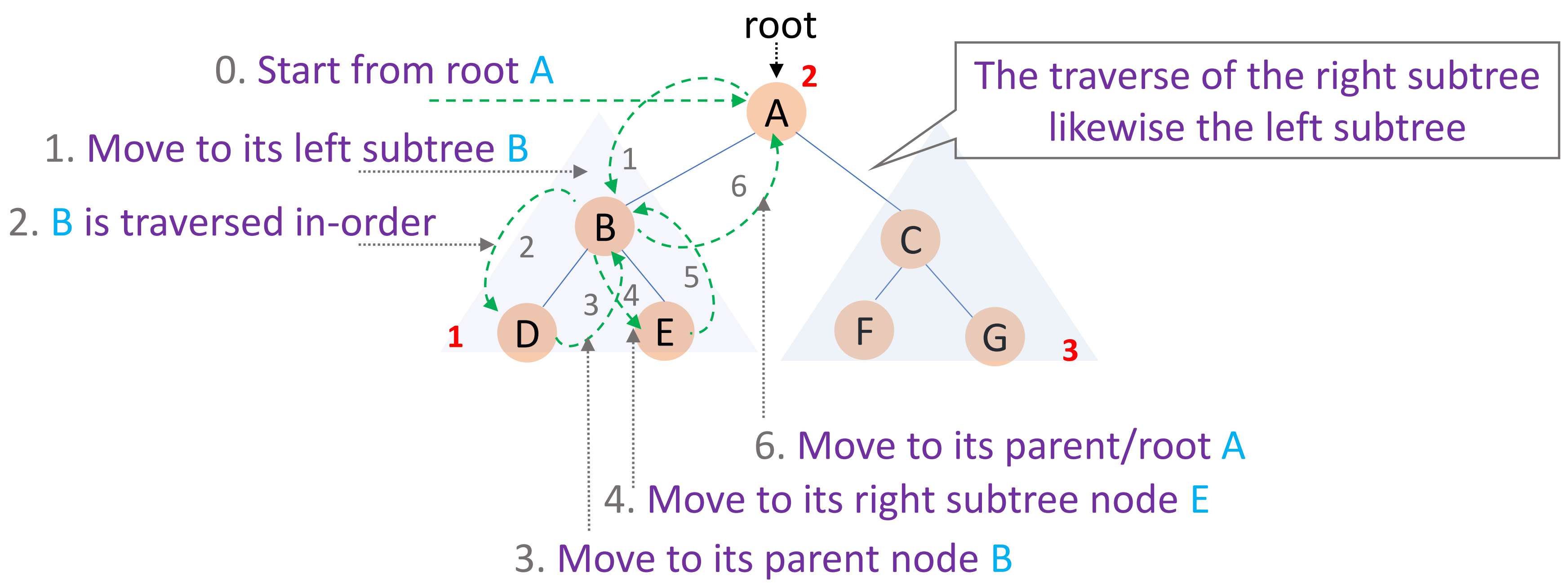
- ✓ **Breadth-First Traversal/Search (BFS):** A traversal or searching algorithm in data structures that explores all the nodes at a current level prior to moving on the next level. There is only one type of BFS:

  1. Level-order Traversal

Depth-First Search (DFS) (1)

- **Inorder Traversal:** In this traversal method, the left subtree is visited first, then the root node and later the right subtree. Every node may represent a subtree itself. The process goes on until all the nodes are visited.

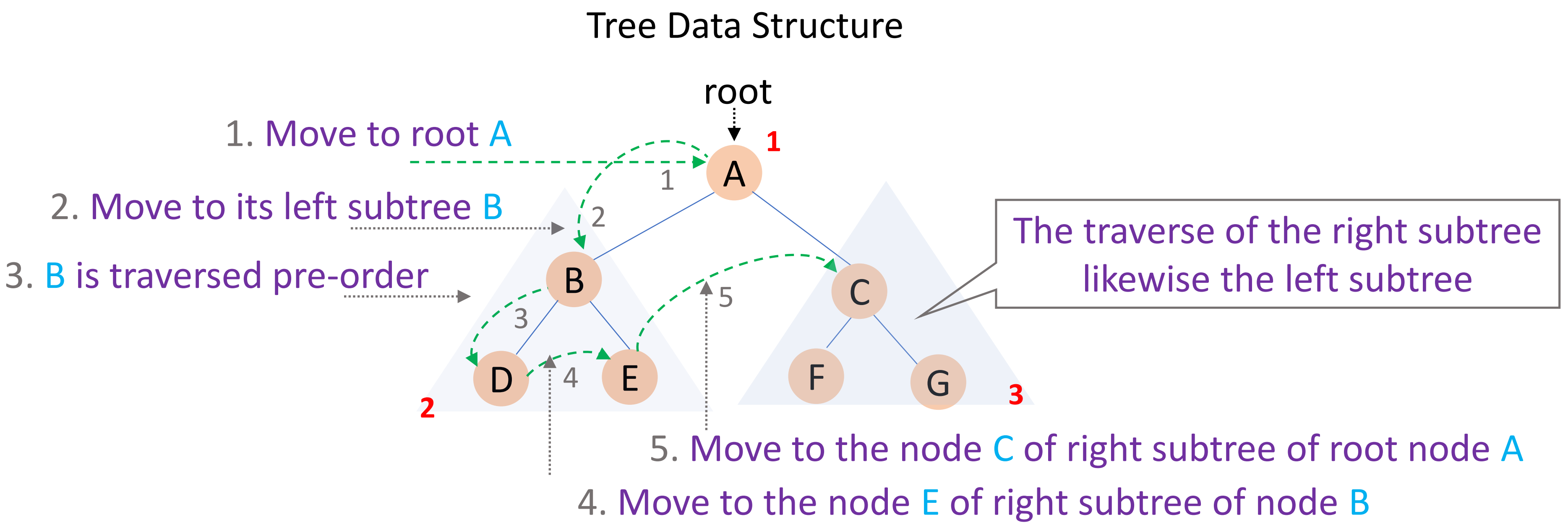
Tree Data Structure



✓ **Output:** The set of the traversed nodes is: {D --> B --> E --> A --> F --> C --> G}

Depth-First Search (DFS) (2)

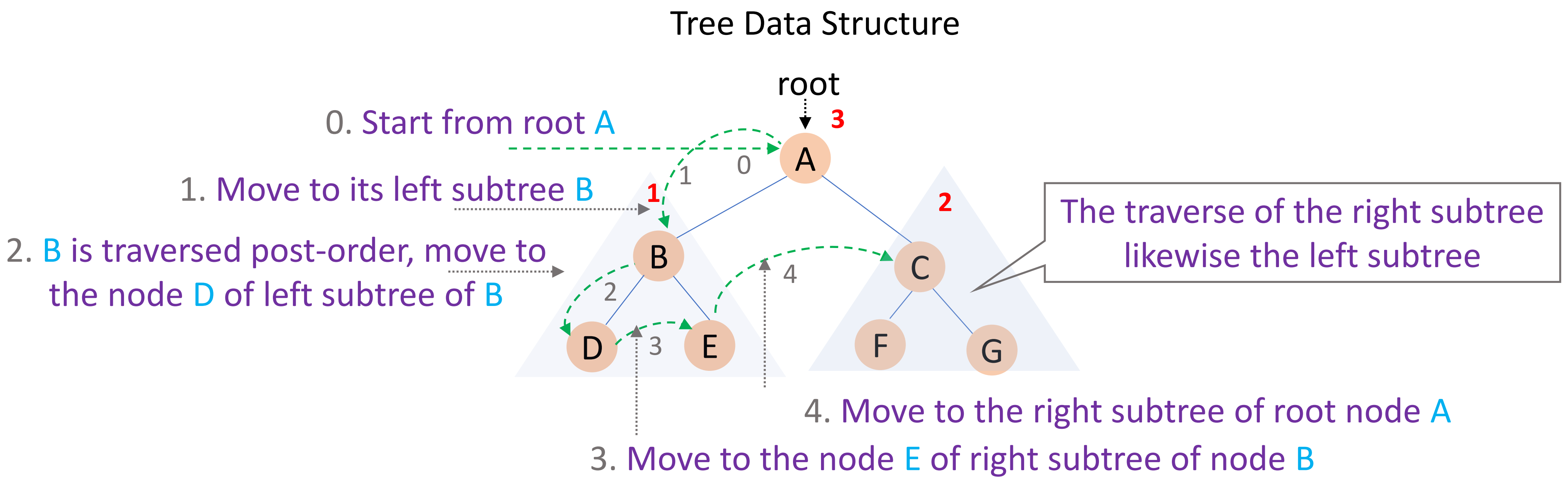
- **Preorder Traversal:** In this traversal method, the root/parent node is visited first, then left subtree and finally the right subtree. Every node may represent a subtree itself. The process goes on until all the nodes are visited.



✓ **Output:** The set of the traversed nodes is: {A --> B --> D --> E --> C --> F --> G}

Depth-First Search (DFS) (3)

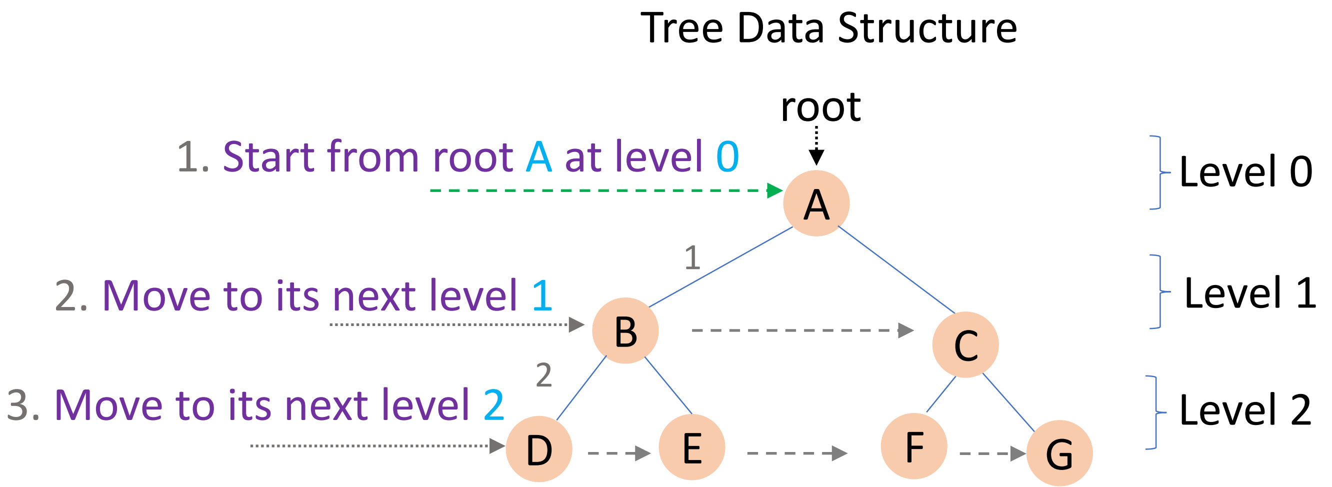
- **Postorder Traversal:** In this traversal method, the left subtree is visited first, then right subtree and finally the root node. Every node may represent a subtree itself. The process goes on until all the nodes are visited.



✓ **Output:** The set of the traversed nodes is: {D --> E --> B --> F --> G --> C --> A}

Breadth-First Search (BFS)

- **Level-order Traversal:** In this traversal method, all the nodes of a tree are visited, one level at a time, from the root to the bottom level.

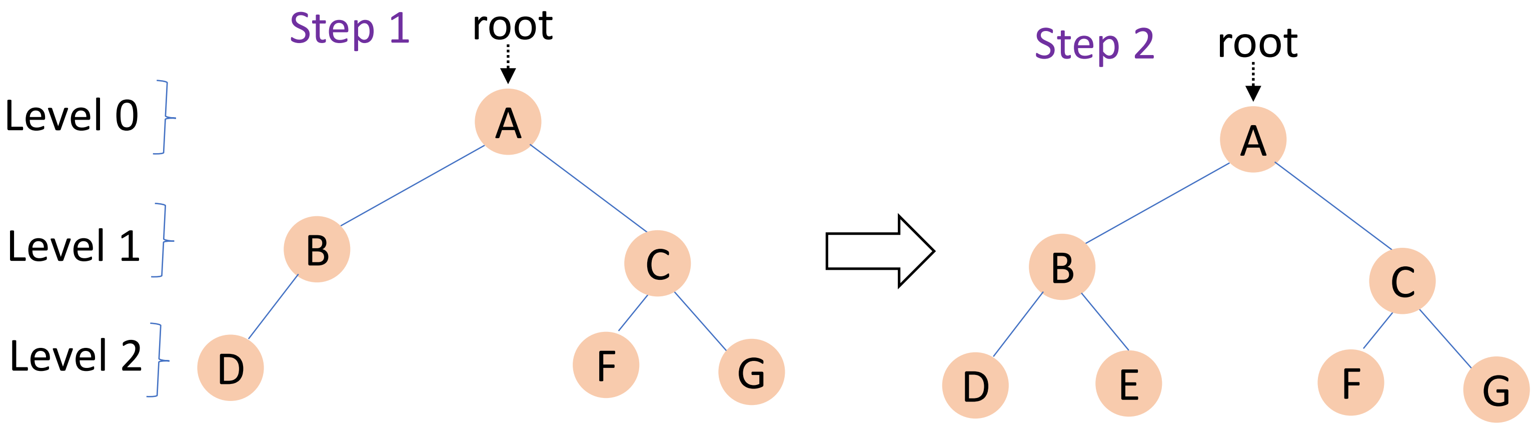


✓ **Output:** The set of the nodes in the tree traversed are {A --> B--> C --> D --> E --> F --> G}

Insertion in Binary Tree

- **Insertion in a binary tree in level order:** Inserts a node into the binary tree at the first position available in level order.

  An example: inserts a node E into the following binary tree.



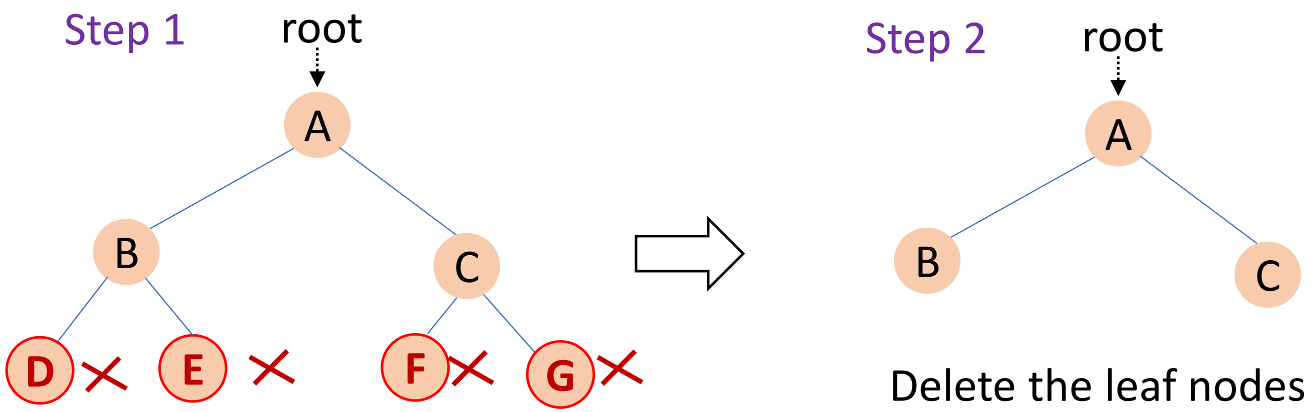
Find the first position available

Insert the node E at the position

Deletion in Binary Tree (1)

- **Deletion a node (a leaf node):** Deletes a leaf node from a binary tree simply.

  An example: deletes the leaf node(s) in the following binary tree.



The leaf nodes to be deleted are D, E, F, and G

Delete the leaf nodes

Deletion in Binary Tree (2)

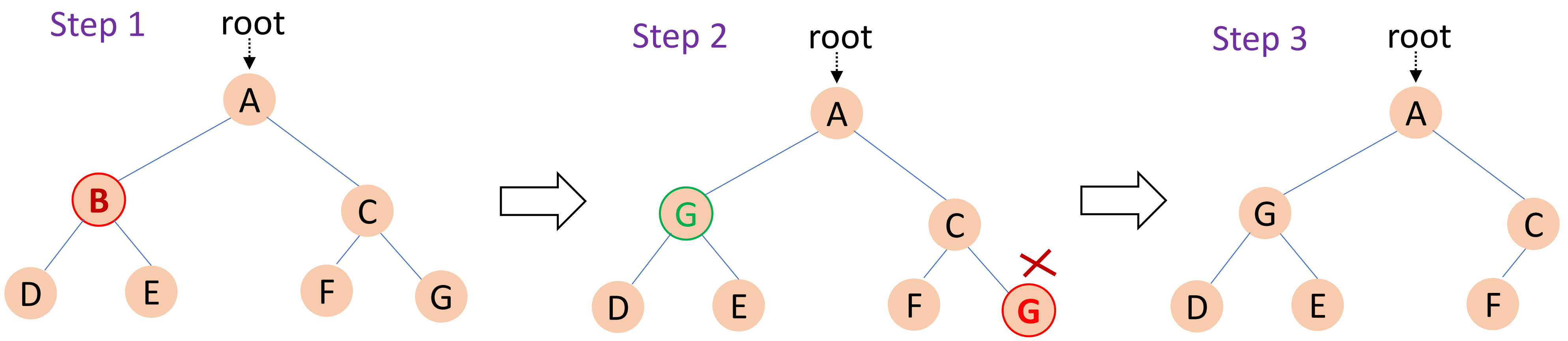
- **Deletion a node (not a leaf node):** Deletes a non leaf node from a binary tree. The process is as follows:

✓ Starting at root, find the deepest and rightmost node in a binary tree and the node which we want to delete.

✓ Replace the deepest rightmost node's data to be deleted one.

✓ Then delete the deepest rightmost node.

An example: deletes the node B in the following binary tree.



A node to be deleted is B

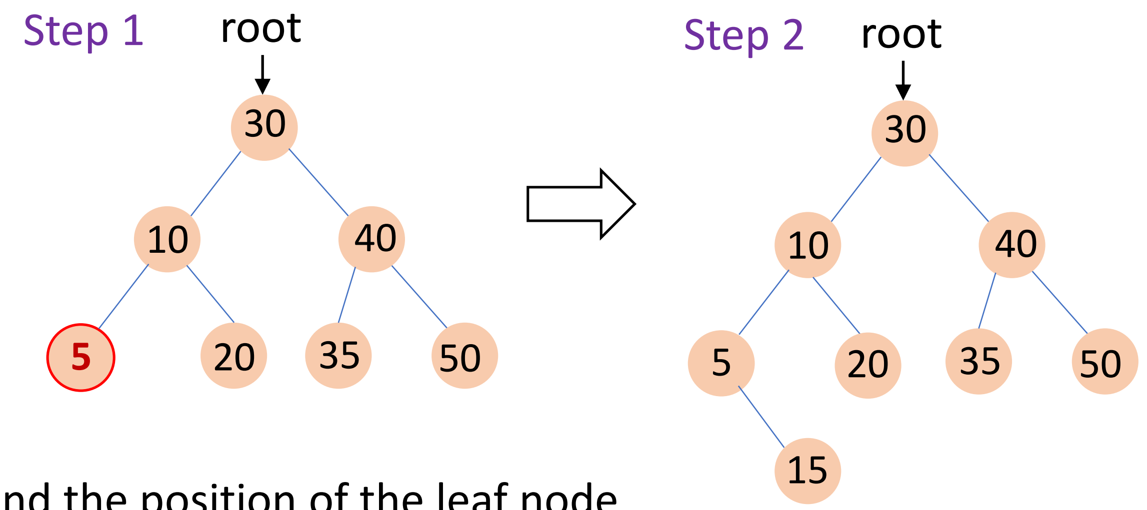
Replacing node B's data with the data of deepest node G

Delete the deepest node G

Insertion in Binary Search Tree (BST)

- **Insertion in a BST:** Inserts a node always into the leaf in a BST with satisfying the condition.

  An example: inserts a node 15 into the following BST.



Find the position of the leaf node where the new node to be inserted

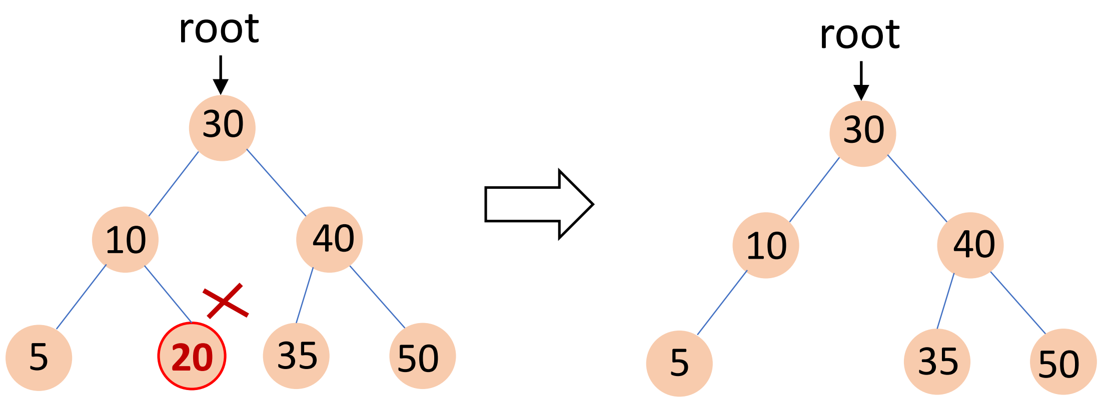
Insert the node into the leaf node

Deletion in Binary Search Tree (BST) (1)

- **Deletion in BST:** Deletes the specified node from a BST. The process is varied on the different types of deletions of the nodes.

✓ **Case I (delete a leaf node):** Deletes the leaf node from the tree simply.

An example: deletes a leaf node 20 from the following BST.



Find the position of the node 20 to be deleted

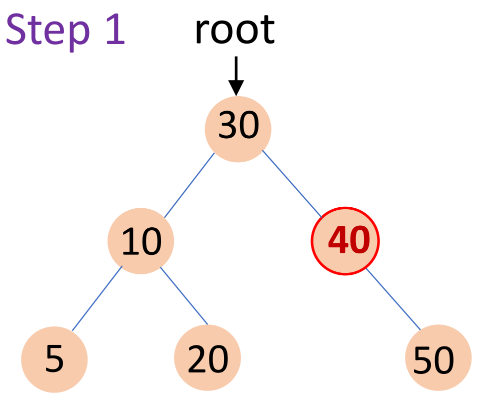
Delete the node

Deletion in Binary Search Tree (BST) (2)

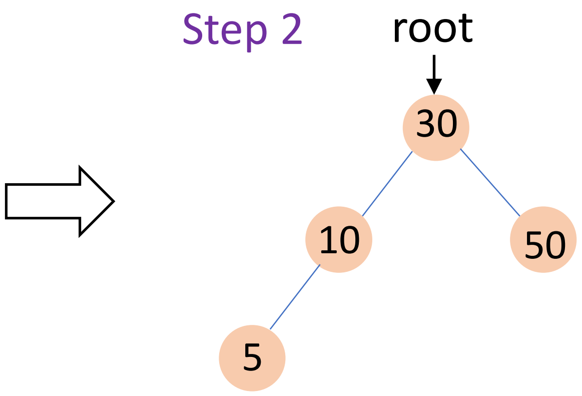
✓ **Case II (delete a node with a single child):** The node to be deleted lies has a single childe node. The process is as follows:

1. Replace that node with its child node.
2. Remove the child node from its original position.

An example: deletes a node has a single child from the following BST.



Find the node(s) has single child, node 40 is to be deleted



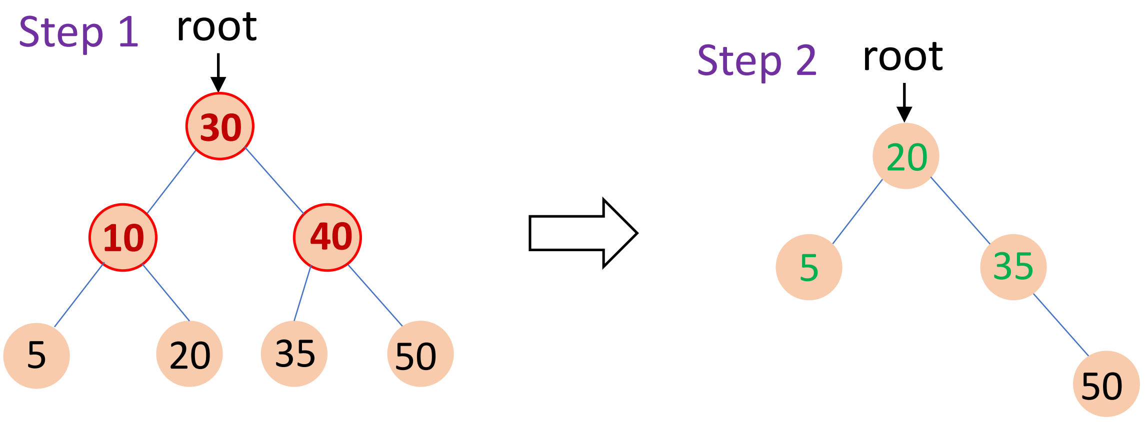
Delete the node 40, replace it by the child node 50, and remove the child node

Deletion in Binary Search Tree (BST) (3)

✓ **Case III (delete a node with two children):** The node to be deleted has two children. The process is as follows:

1. Find the in-order successor of that node.
2. Replace the node with the in-order successor and satisfies either of the following two conditions.
   - minimum element of right subtree
   - maximum element of left subtree
3. Remove the in-order successor from its original position.

An example: delete a node has two children from the following BST.



Find the nodes to be deleted (30, 10 and 40 are to be deleted)

Replace the nodes by its in-order successors, and remove the successors

Binary Search Tree Construction in Python (1)

❑ Insert elements in a Binary Search Tree (BST)

```python
# Binary Search Tree (BST) Construction in Python
#...................................................

# Define a class as BinarySearchTreeNode
class BinarySearchTreeNode:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

# Define a function to insert the values/keys into the BST
def insertNewValue(root,NewValue):
    # If the BST is null/empty, the first value/key as the value in root node
    if root is None:
        root = BinarySearchTreeNode(NewValue)
        return root

    # If the BST is not empty, insert the new value into the tree
    # The value is less than (<) the value in root, insert the value to the left subtree
    if NewValue < root.data:
        root.left = insertNewValue(root.left,NewValue)
    # The value is greater than (>) the value of data in root,insert the value to the right subtree
    else:
        root.right = insertNewValue(root.right, NewValue)
    return root

# Insert the values into the BST
root = insertNewValue(None,6)
insertNewValue(root,4)
insertNewValue(root,3)
insertNewValue(root,5)
insertNewValue(root,8)
insertNewValue(root,7)
insertNewValue(root,9)
```

Binary Search Tree Traversal in Python (2)

❑ Inorder Traversal

```python
# Display the values in a Binary Search Tree
# using Inorder Traversal in Python
#.............................................

# Define a class as BinarySearchTreeNode
class BinarySearchTreeNode:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

# Define a function to insert the values/keys into the BST
def insertNewValue(root,NewValue):
    # If the BST is null/empty, the first value/key as the value in root node
    if root is None:
        root = BinarySearchTreeNode(NewValue)
        return root

    # If the BST is not empty, insert the new value into the tree
    # The value is less than (<) the value in root,
    # insert the value to the left subtree
    if NewValue < root.data:
        root.left = insertNewValue(root.left,NewValue)
    # The value is greater than (>) the value of data in root
    # insert the value to the right subtree
    else:
        root.right = insertNewValue(root.right, NewValue)
    return root
```

In an inorder traversal, the Binary Search Tree is traversed as follows:

1. Traverse the left subtree
2. Visit the parent/root node
3. Traverse the right subtree

```python
# Insert the values into the BST
root = insertNewValue(None,6)
insertNewValue(root,4)
insertNewValue(root,3)
insertNewValue(root,5)
insertNewValue(root,8)
insertNewValue(root,7)
insertNewValue(root,9)

print('\nOutput:')
print("Display the values in the BST using Inorder Traversal")
inorderTraversal(root)
```

```
Output:
Display the values in the BST using Inorder Traversal
3
4
5
6
7
8
9
```

Binary Search Tree Traversal in Python (3)

❑ Preorder Traversal

In a Preorder Traversal, the Binary Search Tree is traversed as follows:

1. Visit the root node
2. Traverse the left subtree
3. Traverse the right subtree

```python
# Display the values in a Binary Search Tree
# using Preorder Traversal in Python
#..................................................

# Define a class as BinarySearchTreeNode
class BinarySearchTreeNode:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

# Define a function to insert the values/keys into the BST
def insertNewValue(root,NewValue):
    # If the BST is null/empty, the first value/key as the value in root node
    if root is None:
        root = BinarySearchTreeNode(NewValue)
        return root

    # If the BST is not empty, insert the new value into the tree
    # The value is less than (<) the value in root,
    # insert the value to the left subtree
    if NewValue < root.data:
        root.left = insertNewValue(root.left,NewValue)
    # The value is greater than (>) the value of data in root
    # insert the value to the right subtree
    else:
        root.right = insertNewValue(root.right, NewValue)
    return root
```

```python
def preorderTraversal(root):
         # If root is None,return
         if root == None:
             return
         # Traverse the root first
         print(root.data)
         # Traverse the node in the left subtree
         preorderTraversal(root.left)
         # Traverse the node in the right subtree
         preorderTraversal(root.right)

# Insert the values into the BST
root = insertNewValue(None,6)
insertNewValue(root,4)
insertNewValue(root,3)
insertNewValue(root,5)
insertNewValue(root,8)
insertNewValue(root,7)
insertNewValue(root,9)

print('\nOutput:')
print("Display the values in the BST using Preorder Traversal")
preorderTraversal(root)
```

```
Output:
Display the values in the BST using Preorder Traversal
6
4
3
5
8
```

Binary Search Tree Traversal in Python (4)

In a Postorder Traversal, the Binary Search Tree is traversed as follows:

1. Traverse the left subtree
2. Traverse the right subtree
3. Visit the root node

❑ Postorder Traversal

```python
# Display the values in a Binary Search Tree
# using Postorder Traversal in Python
#.................................................

# Define a class as BinarySearchTreeNode
class BinarySearchTreeNode:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None

# Define a function to insert the values/keys into the BST
def insertNewValue(root,NewValue):
    # If the BST is null/empty, the first value/key as the value in root node
    if root is None:
        root = BinarySearchTreeNode(NewValue)
        return root

    # If the BST is not empty, insert the new value into the tree
    # The value is less than (<) the value in root,
    # insert the value to the left subtree
    if NewValue < root.data:
        root.left = insertNewValue(root.left,NewValue)
    # The value is greater than (>) the value of data in root
    # insert the value to the right subtree
    else:
        root.right = insertNewValue(root.right, NewValue)
    return root
```

```python
def postorderTraversal(root):
        # If root is None,return
        if root == None:
            return
        # Traverse the node in the left subtree
        preorderTraversal(root.left)
        # Traverse the node in the right subtree
        preorderTraversal(root.right)
        # Traverse the root node
        print(root.data)

# Insert the values into the BST
root = insertNewValue(None,6)
insertNewValue(root,4)
insertNewValue(root,3)
insertNewValue(root,5)
insertNewValue(root,8)
insertNewValue(root,7)
insertNewValue(root,9)

print('\nOutput:')
print("Display the values in the BST using Postorder Traversal")
postorderTraversal(root)
```

```
Output:
Display the values in the BST using Postorder Traversal
4
3
5
8
7
9
6
```