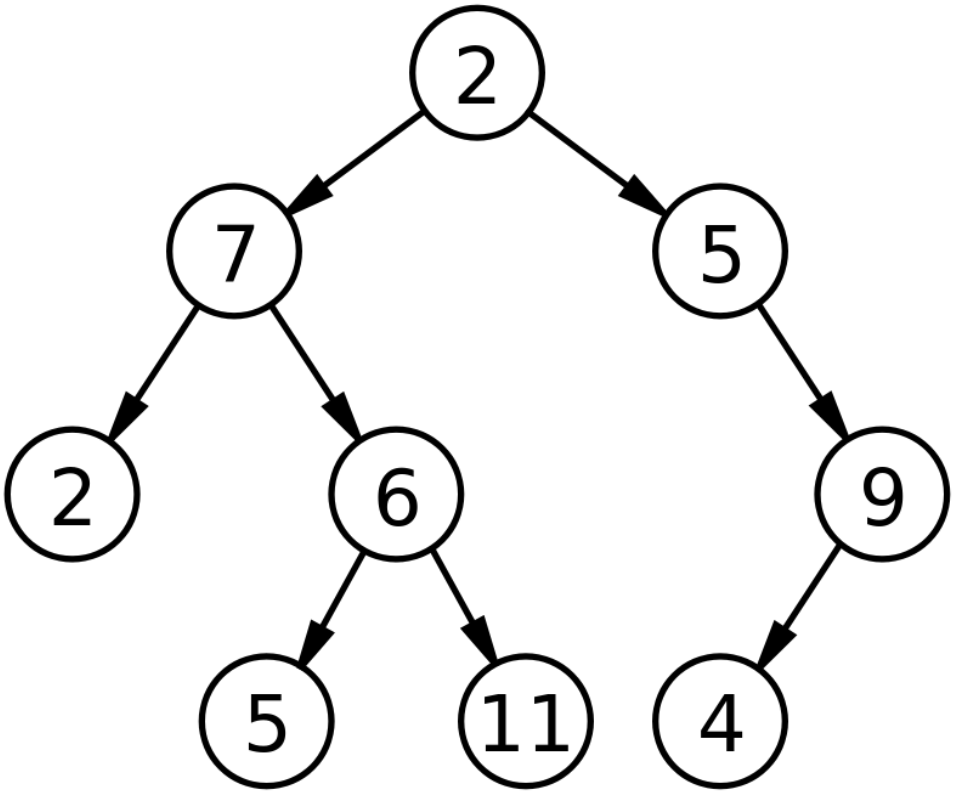
Data Structures

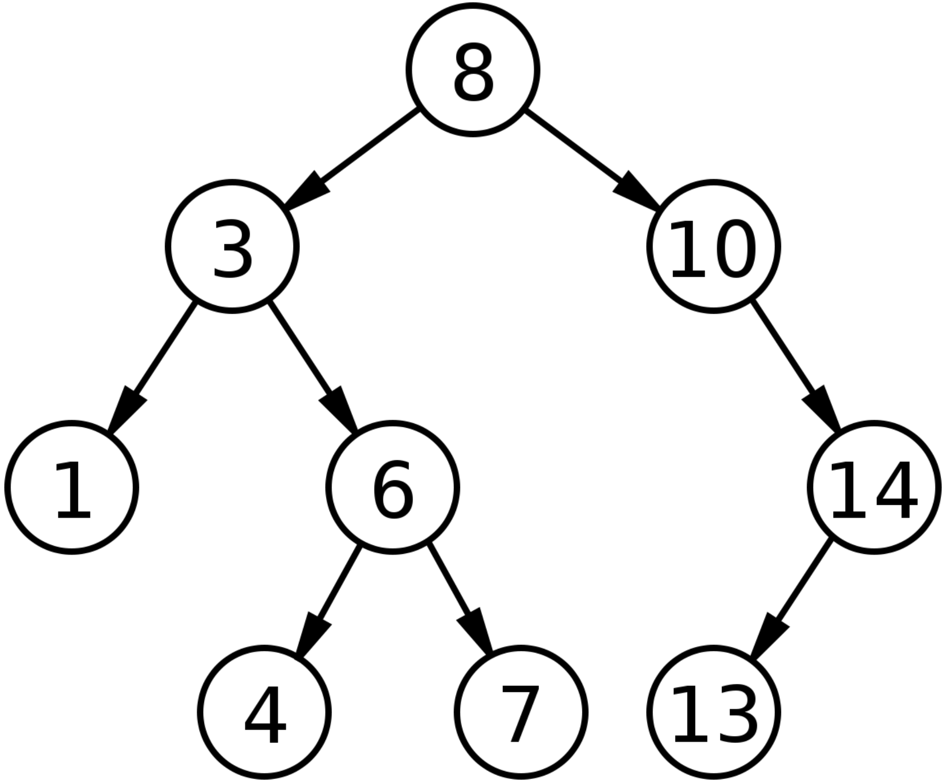
Lecture 12
Binary Search Tree

Credit: Prantik Paul, Lecturer, BRAC University

Binary Tree



Binary Search Tree



Binary Search Tree - Restrictions

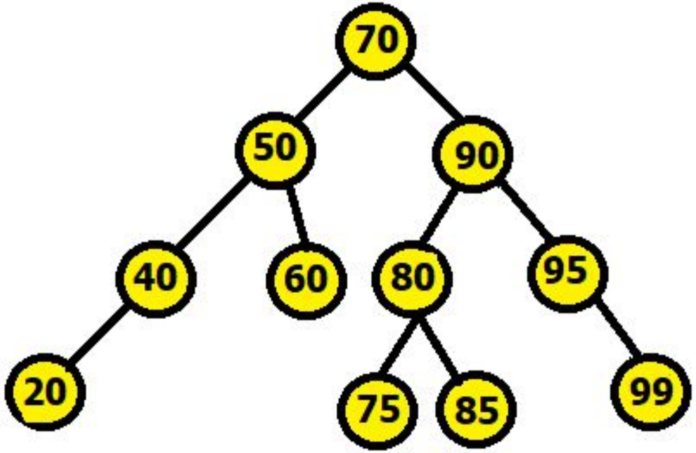
Elements of Left Subtree < Element of Parent

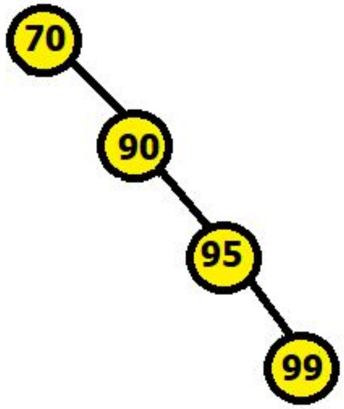
Elements of Right Subtree> Element of Parent

Left and Right Subtree also BST’s

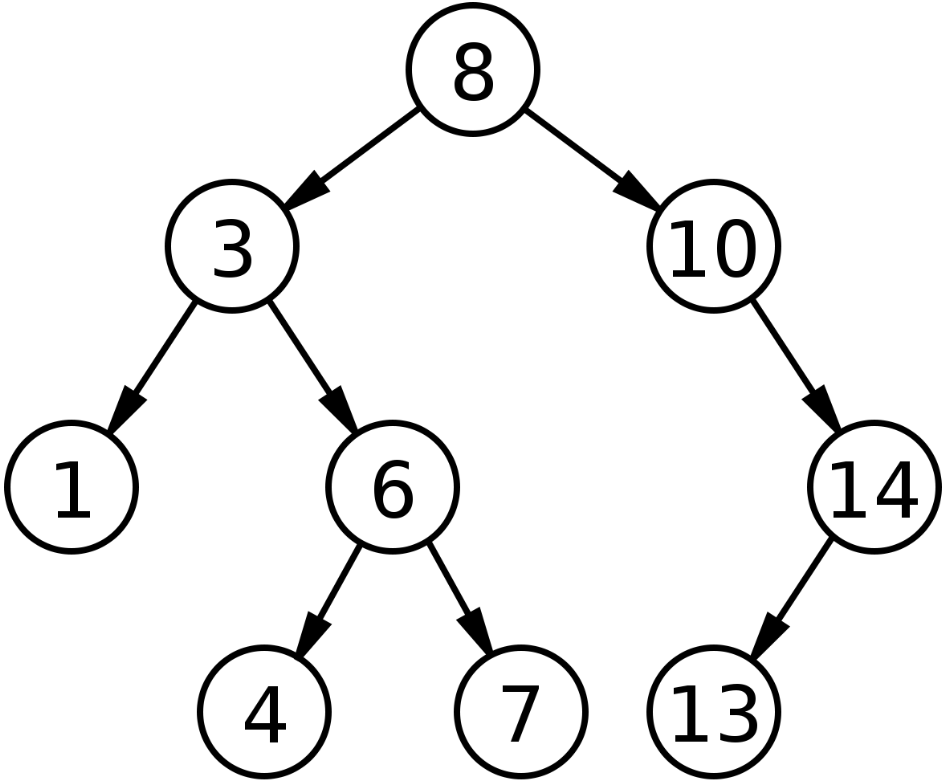
No Duplicate Values

Binary Search Tree





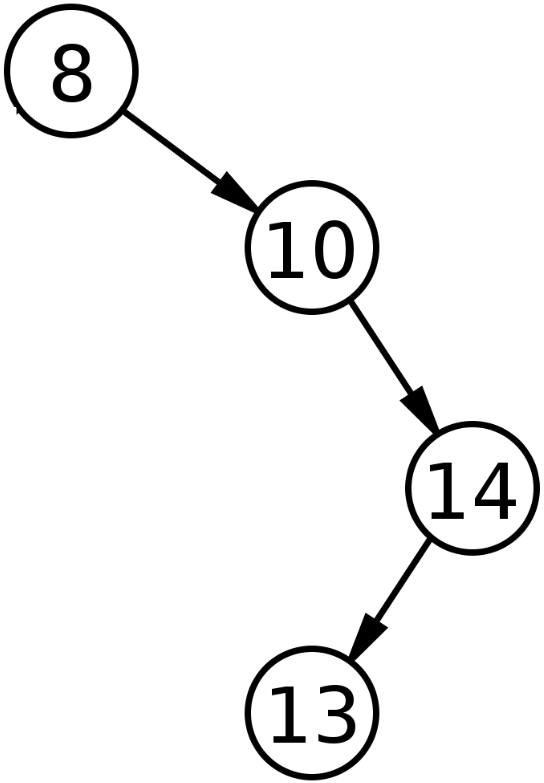
Binary Search Tree - Why to use



Search 7

Search 5

Binary Search Tree - Why to use



Search 12

Must be Balanced

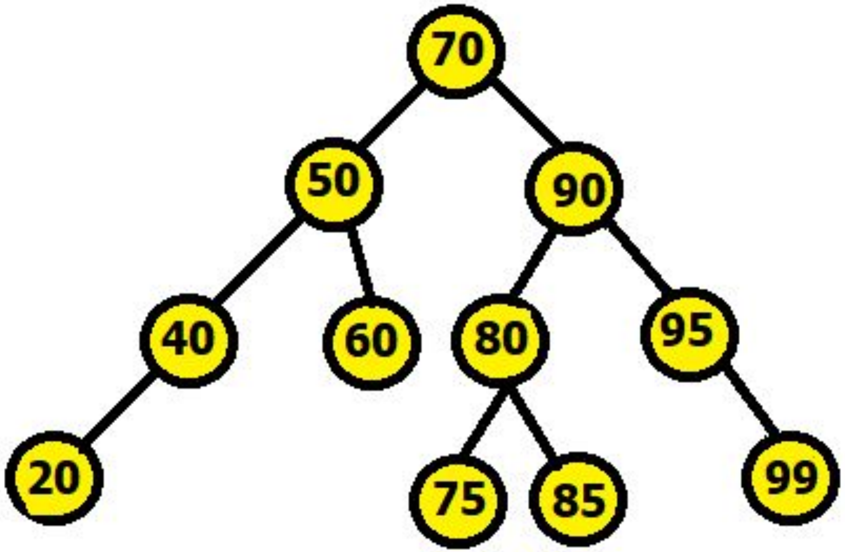
Binary Search Tree - Creation

70, 50, 40, 90, 20, 95, 99, 80, 85, 75

Binary Search Tree - Creation

70, 50, 40, 90, 20, 95, 99, 80, 85, 75

Binary Search Tree - Insertion



Insert 10

Insert 5

Must Make Balanced

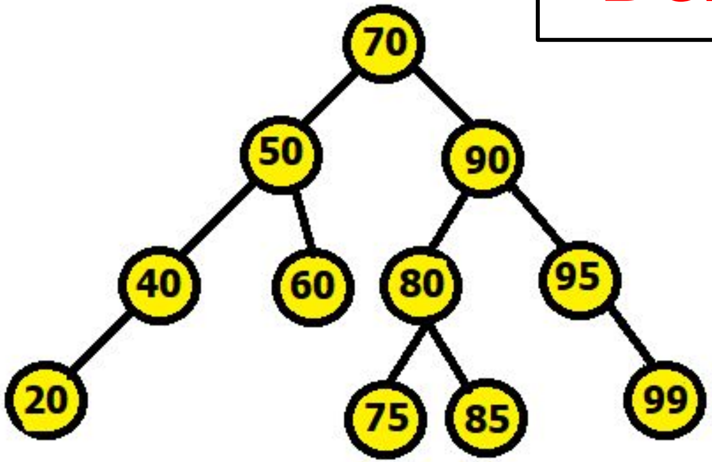
Binary Search Tree - Insert Node

```
def addNode(root, i): #Adding Nodes
  if i<root.elem and root.left==None:
    n= Node(i)
    root.left= n
  elif i>root.elem and root.right==None:
    n= Node(i)
    root.right= n
  if i<root.elem and root.left!=None:
    addNode(root.left, i)
  elif i>root.elem and root.right!=None:
    addNode(root.right, i)
```

Binary Search Tree - Deletion

Case 1: No Subtree or Children

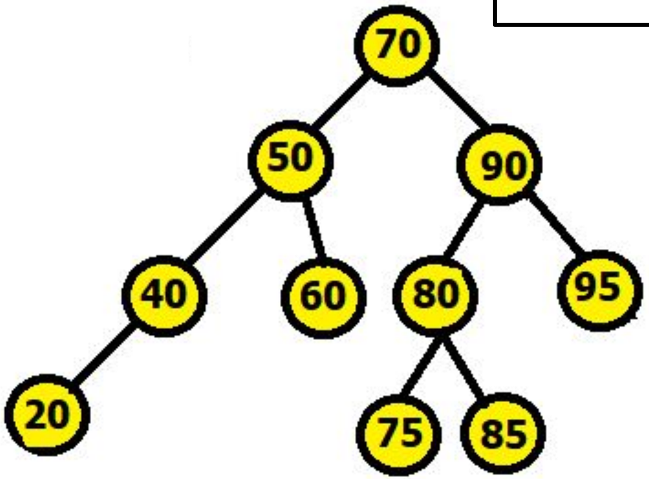
Delete 99



Binary Search Tree - Deletion

Case 2: Only 1 Subtree or Child

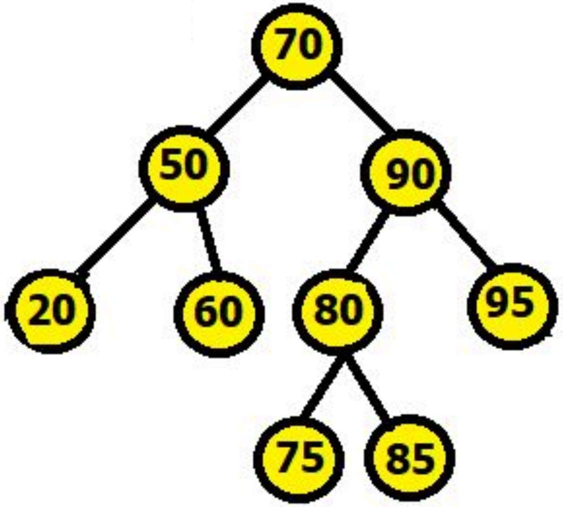
Delete 40



Binary Search Tree - Deletion

Case 3: 2 Subtrees or Children (Replace by Inorder Successor)

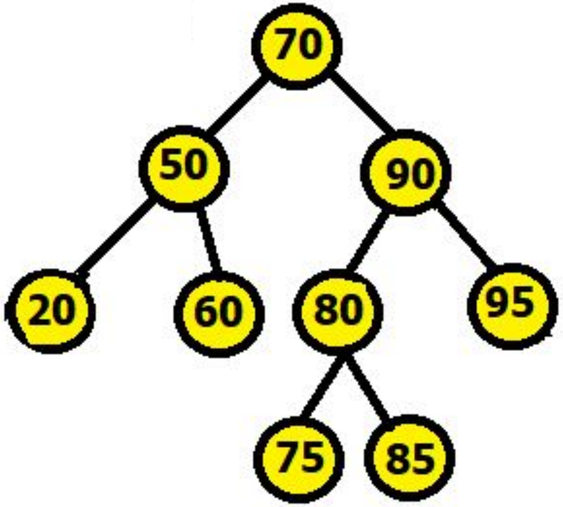
Delete 90



Binary Search Tree - Deletion

Case 3: 2 Subtrees or Children (Replace by Inorder Predecessor)

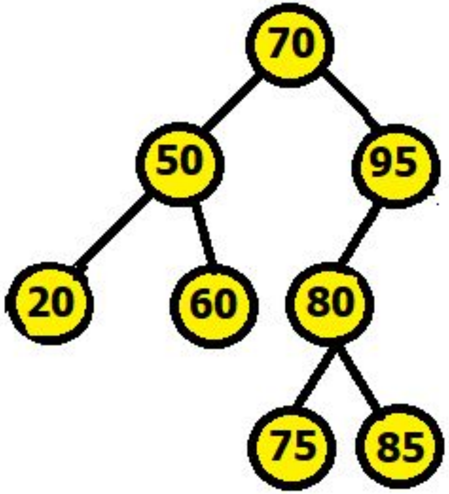
Delete 90



Binary Search Tree - Deletion

Case 3: 2 Subtrees or Children (Replace by Inorder Successor)

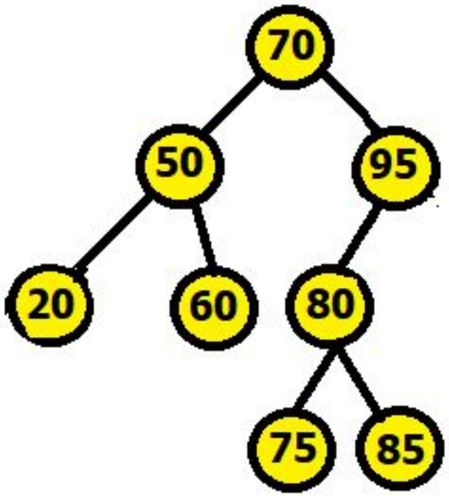
Delete 70



Binary Search Tree - Deletion

Case 3: 2 Subtrees or Children (Replace by Inorder Predecessor)

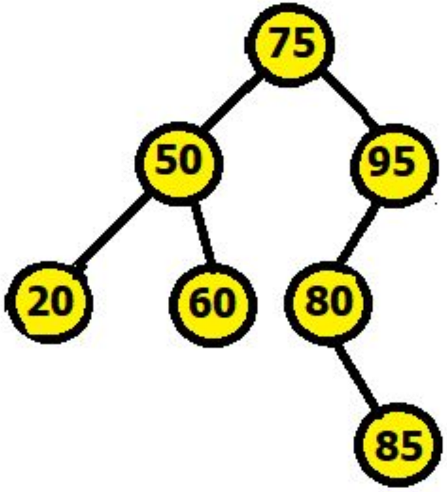
Delete 70



Binary Search Tree - Deletion

Case 3: 2 Subtrees or Children (Replace by Inorder Successor)

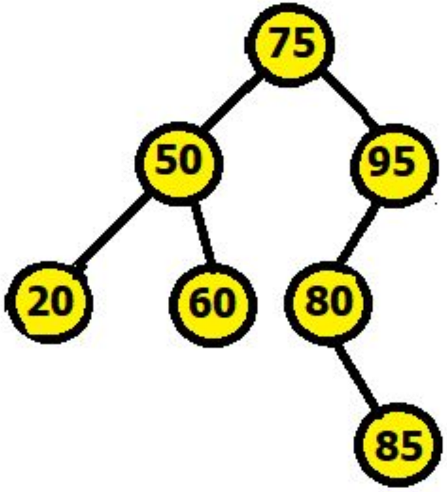
Delete 75



Binary Search Tree - Deletion

Case 3: 2 Subtrees or Children (Replace by Inorder Predecessor)

Delete 75



Binary Search Tree - Delete Node

```
def minValueNode(node):
    current = node
    while(current.left is not None): # loop down to find the leftmost leaf
        current = current.left
    return current
```

Binary Search Tree - Delete Node

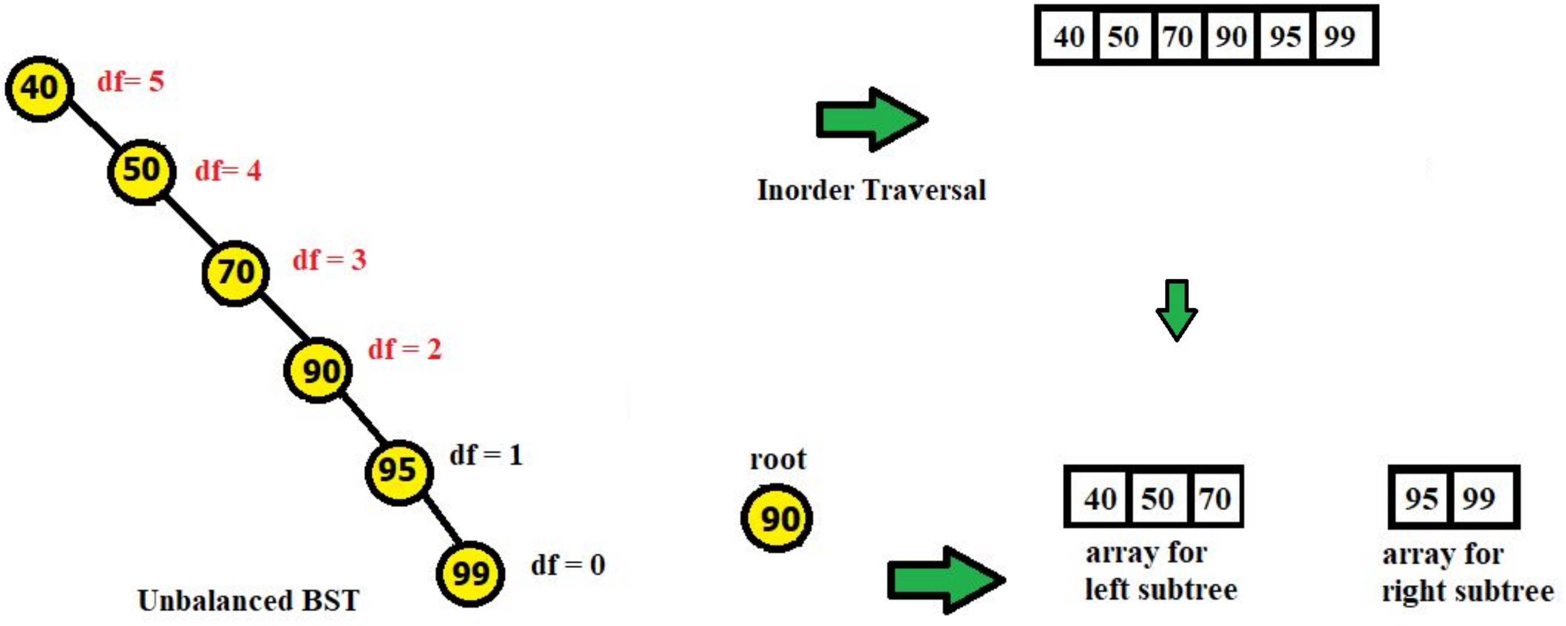
```
def deleteNode(root, key):
    if root is None:
        return root
    # If the key to be deleted is smaller than the root's key then it lies in  left subtree
    if key < root.elem:
        root.left = deleteNode(root.left, key)
    # If the kye to be delete is greater than the root's key then it lies in right subtree
    elif(key > root.elem):
        root.right = deleteNode(root.right, key)
```

Binary Search Tree - Delete Node

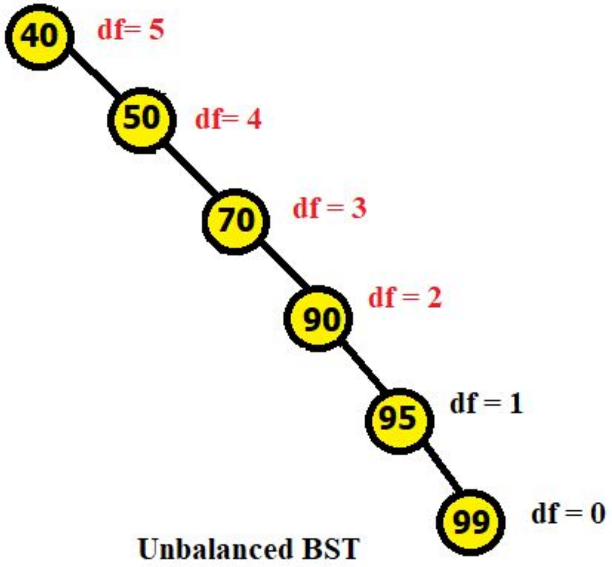
```
# If key is same as root's key, then this is the node to be deleted
else:
    # Node with only one child or no child
    if root.left is None:
        temp = root.right
        root = None
        return temp
    elif root.right is None:
        temp = root.left
        root = None
        return temp
    # Node with two children:
    # Get the inorder successor (smallest in the right subtree)
    temp = minValueNode(root.right)
    # Copy the inorder successor's content to this node
    root.key = temp.elem
    root.right = deleteNode(root.right, temp.elem) # Delete the inorder successor
return root
```

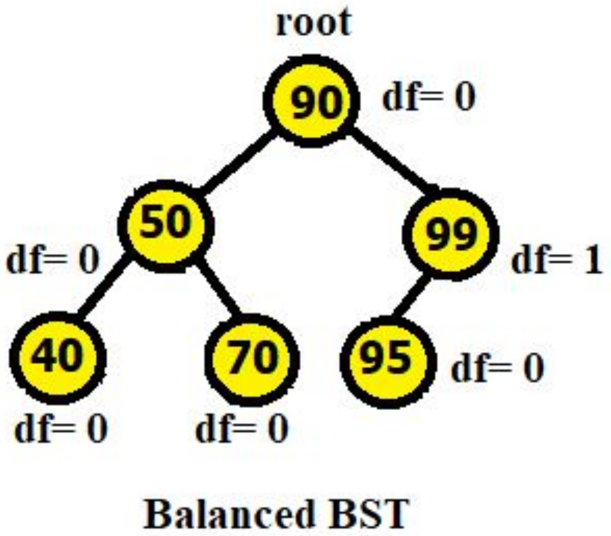
Binary Search Tree - Balancing

Binary Search Tree - Balancing



Binary Search Tree - Balancing





Binary Search Tree - Balance Tree

```
#Appends into a list with tree nodes using inorder traversal
def pushTreeNodes(root, arr):
    if root is None:
        return
    pushTreeNodes(root.left, arr)
    arr.append(root)
    pushTreeNodes(root.right, arr)
```

Binary Search Tree - Balance Tree

```
# Recursive function to construct a height-balanced BST from
# given nodes in sorted order
def buildBalancedBST(arr, start, end):
    if start > end:
        return None
    mid = (start + end) // 2 # find the middle index
    root = arr[mid] # The root node will be a node present at the mid-index
    # recursively construct left and right subtree
    root.left = buildBalancedBST(arr, start, mid - 1)
    root.right = buildBalancedBST(arr, mid + 1, end)
    return root
```

Binary Search Tree - Search Node in Tree

Search a node in a BST

DIY