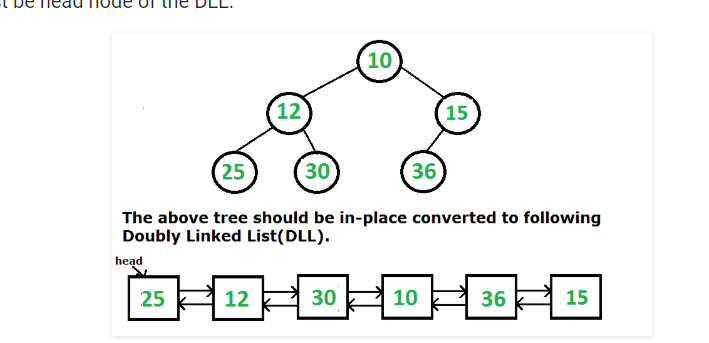
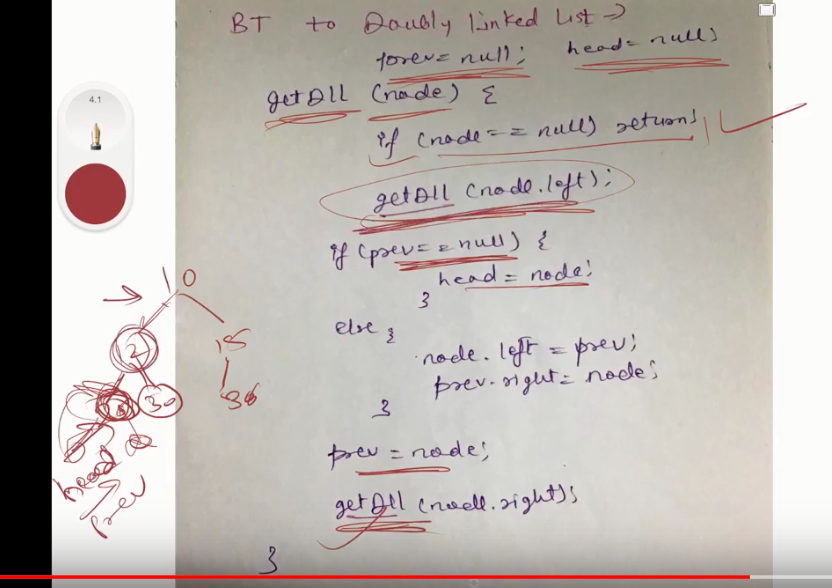
Convert a given Binary Tree to Doubly Linked List | Set 1

**We will do inorder traversal to convert it into doubly linked list.**







# height of a Binary Tree

# 

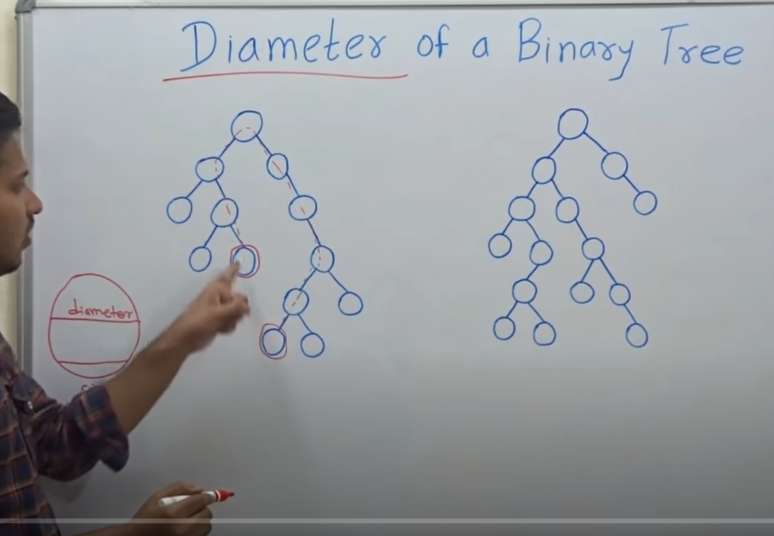
# 

# 

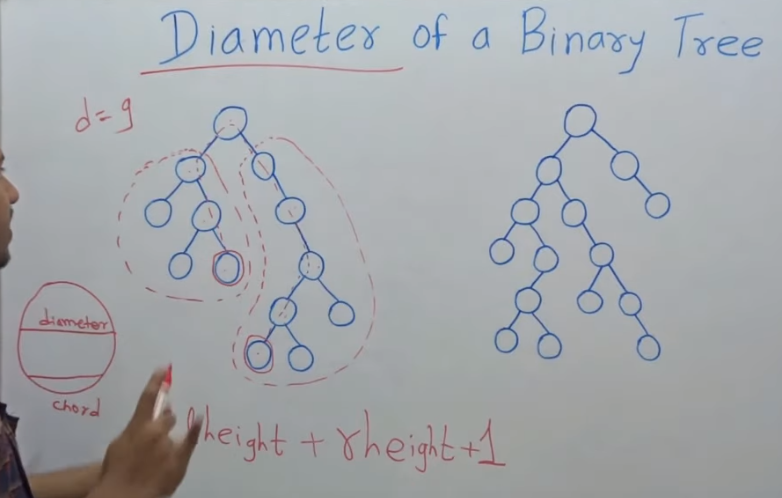
# 

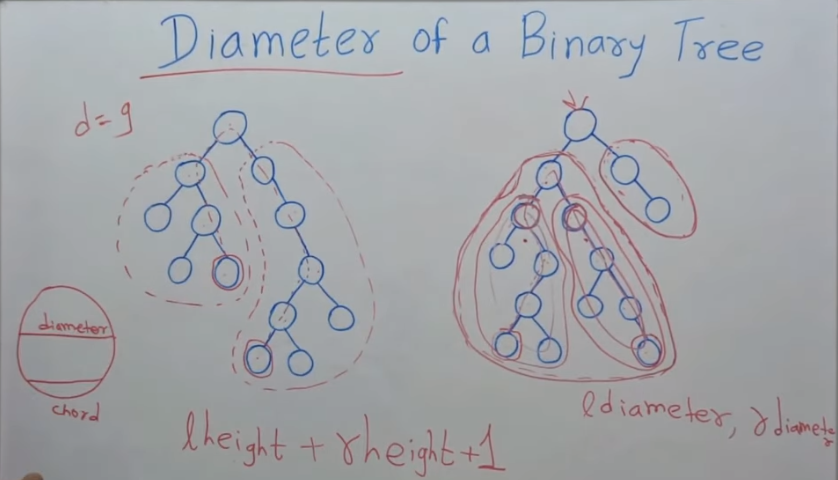
# Diameter of a Binary Tree

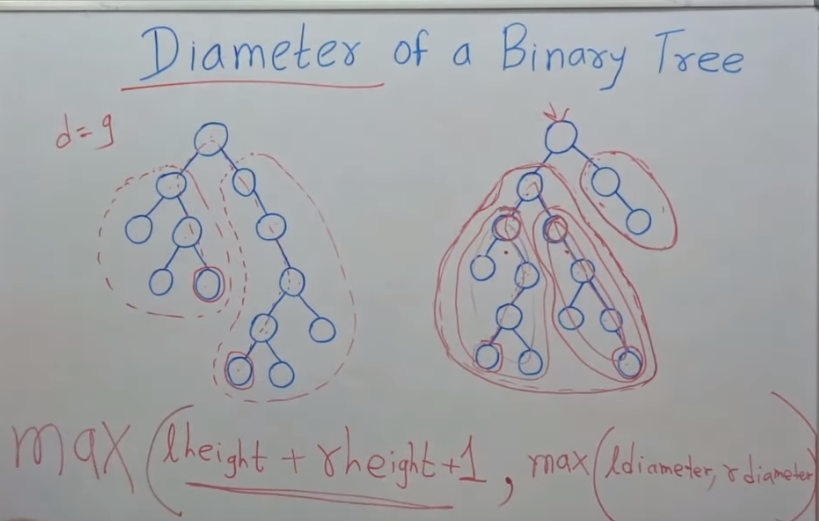
Number of nodes of the longest path of the binary tree.



Here the redlines indicates the diameter of the binary tree.







# Python program to find the diameter of binary tree

# A binary tree node

class Node:

    # Constructor to create a new node

    def \_\_init\_\_(self, data):

        self.data = data

        self.left = None

        self.right = None

"""

The function Compute the "height" of a tree. Height is the

number f nodes along the longest path from the root node

down to the farthest leaf node.

"""

def height(node):

    # Base Case : Tree is empty

    if node is None:

        return 0 ;

    # If tree is not empty then height = 1 + max of left

    # height and right heights

    return 1 + max(height(node.left) ,height(node.right))

# Function to get the diamtere of a binary tree

def diameter(root):

    # Base Case when tree is empty

    if root is None:

        return 0;

    # Get the height of left and right sub-trees

    lheight = height(root.left)

    rheight = height(root.right)

    # Get the diameter of left and irgh sub-trees

    ldiameter = diameter(root.left)

    rdiameter = diameter(root.right)

    # Return max of the following tree:

    # 1) Diameter of left subtree

    # 2) Diameter of right subtree

    # 3) Height of left subtree + height of right subtree +1

    return max(lheight + rheight + 1, max(ldiameter, rdiameter))

#here the left condition when the diameter pass through the root node.Here we are passing the plus condition means we are adding the height of the left subtree and height of right subree and adding 1 for the root node.

# Driver program to test above functions

"""

Constructed binary tree is

            1

          /   \

        2      3

      /  \

    4     5

"""

root = Node(1)

root.left = Node(2)

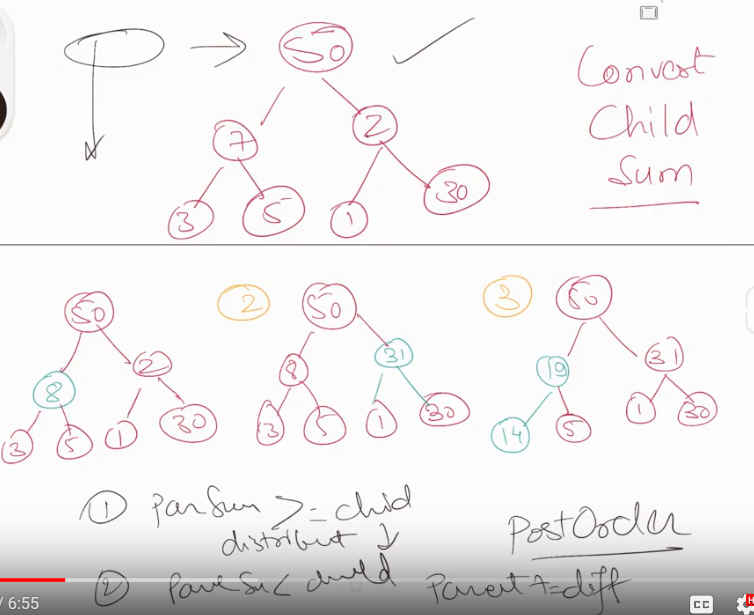
root.right = Node(3)

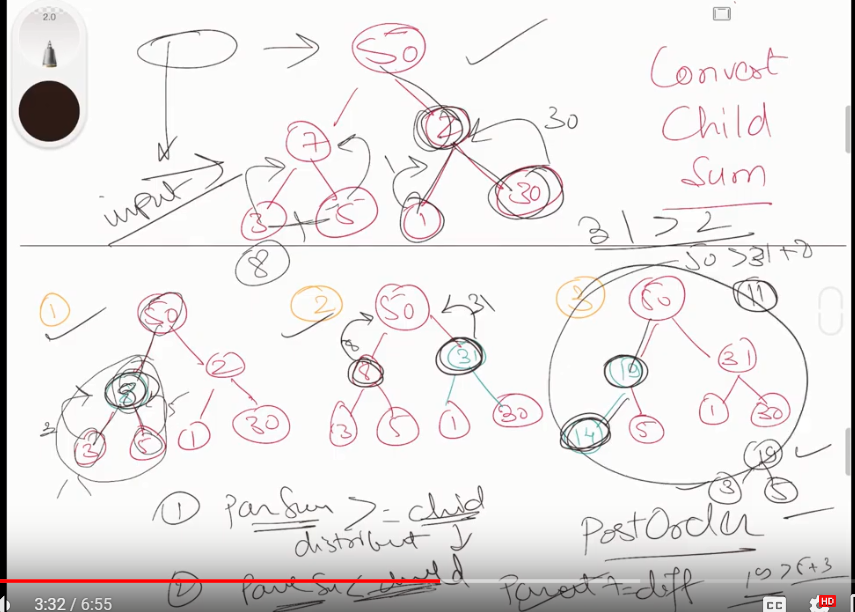
root.left.left = Node(4)

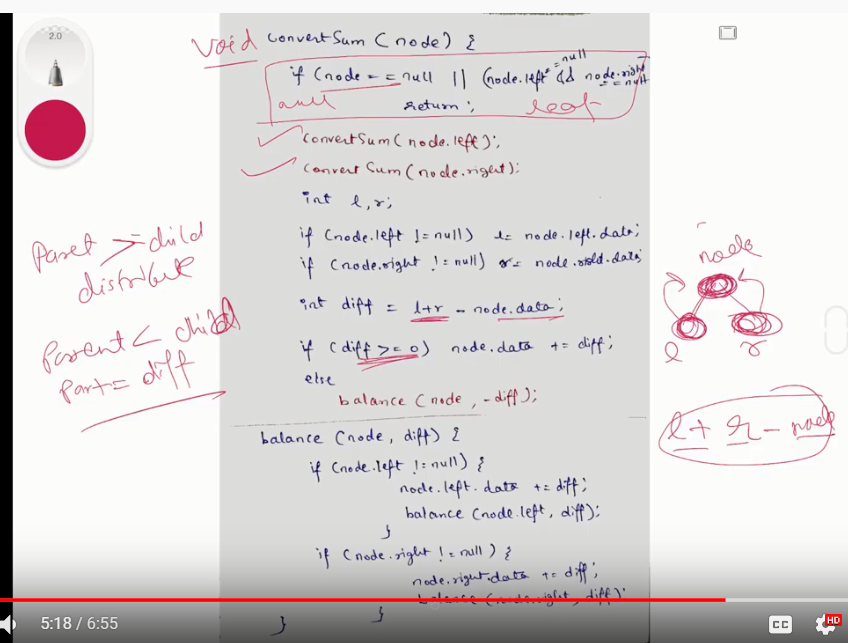
root.left.right = Node(5)

print "Diameter of given binary tree is %d" %(diameter(root))

# Convert an arbitrary Binary Tree to a tree that holds Children Sum Property







If diff is 0 then nothing needs to be done.

If diff > 0 ( node’s data is smaller than node’s children sum) increment the node’s data by diff.

If diff < 0 (node’s data is greater than the node's children sum) then increment one child’s data. We can choose to increment either left or right child if they both are not NULL. Let us always first increment the left child. Incrementing a child changes the subtree’s children sum property so we need to change left subtree also. So we recursively increment the left child. If left child is empty then we recursively call increment() for right child.

# Program to convert an aribitary binary tree

# to a tree that holds children sum property

# Helper function that allocates a new

# node with the given data and None

# left and right poers.

class newNode:

    # Construct to create a new node

    def \_\_init\_\_(self, key):

        self.data = key

        self.left = None

        self.right = None

# This function changes a tree to

# hold children sum property

def convertTree(node):

    left\_data = 0

    right\_data = 0

    diff=0

    # If tree is empty or it's a

    # leaf node then return true

    if (node == None or (node.left == None and

                         node.right == None)):

        return

    else:

        """ convert left and right subtrees """

        convertTree(node.left)

        convertTree(node.right)

    """ If left child is not present then 0

    is used as data of left child """

    if (node.left != None):

        left\_data = node.left.data

    """ If right child is not present then 0

    is used as data of right child """

    if (node.right != None):

        right\_data = node.right.data

    """ get the diff of node's data

        and children sum """

    diff = left\_data + right\_data - node.data

    """ If node's children sum is greater

        than the node's data """

    if (diff > 0):

        node.data = node.data + diff

    """ THIS IS TRICKY -. If node's data is

    greater than children sum, then increment

    subtree by diff """

    if (diff < 0):

        increment(node, -diff) # -diff is used to

                               # make diff positive

""" This function is used to increment

    subtree by diff """

def increment(node, diff):

    """ IF left child is not None

        then increment it """

    if(node.left != None):

        node.left.data = node.left.data + diff

        # Recursively call to fix the

        # descendants of node.left

        increment(node.left, diff)

    elif(node.right != None): # Else increment right child

        node.right.data = node.right.data + diff

        # Recursively call to fix the

        # descendants of node.right

        increment(node.right, diff)

""" Given a binary tree, printInorder()

prints out its inorder traversal"""

def printInorder(node):

    if (node == None):

        return

    """ first recur on left child """

    printInorder(node.left)

    """ then print the data of node """

    print(node.data,end=" ")

    """ now recur on right child """

    printInorder(node.right)

# Driver Code

if \_\_name\_\_ == '\_\_main\_\_':

    root = newNode(50)

    root.left     = newNode(7)

    root.right     = newNode(2)

    root.left.left = newNode(3)

    root.left.right = newNode(5)

    root.right.left = newNode(1)

    root.right.right = newNode(30)

    print("Inorder traversal before conversion")

    printInorder(root)

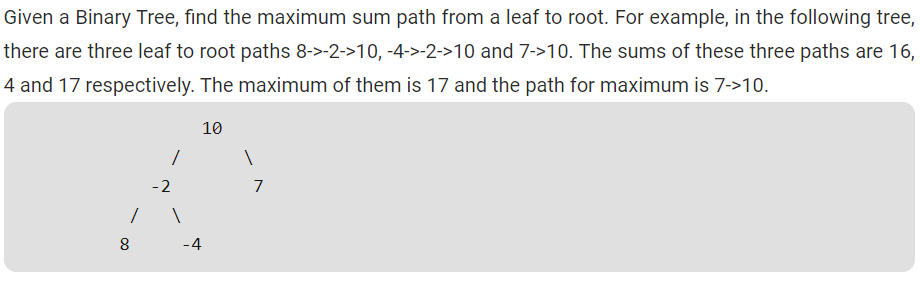
    convertTree(root)

    print("\nInorder traversal after conversion ")

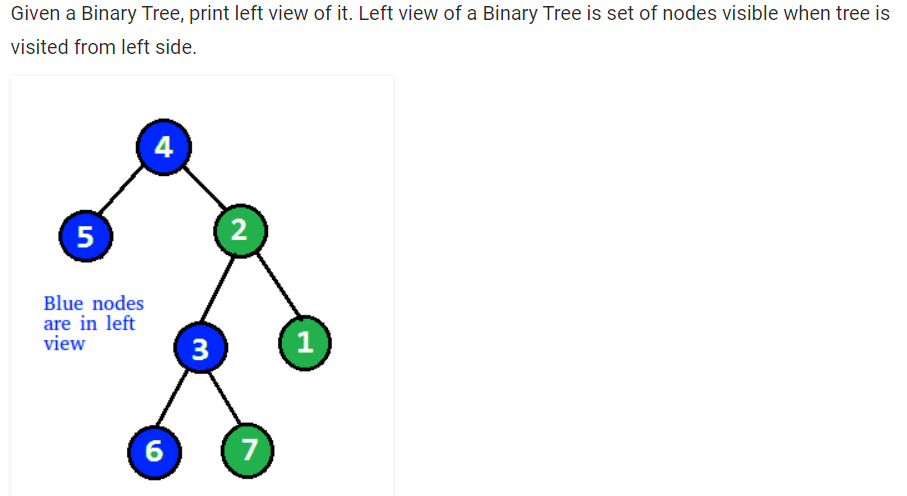
    printInorder(root)

# This code is contributed by

# Find the maximum sum leaf to root path in a Binary Tree



# Iterative Method To Print Left View of a Binary Tree



The idea is to do level order traversal of the Tree using a queue and print the first node at each level.

While doing level order traversal, after traversing all node at each level, push a NULL delimiter to mark the end of the current level. So, do the level order traversal of the tree. Print the first node at each level in the tree and push the children of all nodes at each level in the queue until a NULL delimiter is encountered.

# Python3 program to print the

# left view of Binary Tree

# Binary Tree Node

""" utility that allocates a newNode

with the given key """

class newNode:

    # Construct to create a newNode

    def \_\_init\_\_(self, key):

        self.data = key

        self.left = None

        self.right = None

        self.hd=0

# Utility function to print the left

# view of the binary tree

def leftViewUtil(root, q) :

    if (root == None) :

        return

    # append root

    q.append(root)

    # Delimiter

    q.append(None)

    while (len(q)):

        temp = q[0]

        if (temp):

            # Prints first node of each level

            print(temp.data, end = " ")

            # append children of all nodes

            # at current level

            while (q[0] != None) :

                temp = q[0]

                # If left child is present

                # append into queue

                if (temp.left) :

                    q.append(temp.left)

                # If right child is present

                # append into queue

                if (temp.right) :

                    q.append(temp.right)

                # Pop the current node

                q.pop(0)

            # append delimiter

            # for the next level

            q.append(None)

        # Pop the delimiter of

        # the previous level

        q.pop(0)

# Fucntion to print the leftView

# of Binary Tree

def leftView(root):

    # Queue to store all

    # the nodes of the tree

    q = []

    leftViewUtil(root, q)

# Driver Code

if \_\_name\_\_ == '\_\_main\_\_':

    root = newNode(10)

    root.left = newNode(12)

    root.right = newNode(3)

    root.left.right = newNode(4)

    root.right.left = newNode(5)

    root.right.left.right = newNode(6)

    root.right.left.right.left = newNode(18)

    root.right.left.right.right = newNode(7)

    leftView(root)