Check if a binary tree is balanced

class Solution:

# @param root, a tree node

# @return a boolean

def isBalanced(self, root):

if root == None: return True

else:

maxHeight = self.isBalance(root)

if maxHeight != False:

return True

else: return False

def isBalance(self, root):

if root == None:

return 0

left = self.isBalance(root.left)

if left is False: return False

right = self.isBalance(root.right)

if right is False: return False

if abs(left - right) > 1: return False

else:

return max(left, right) + 1

Convert Sorted Array to Binary Search Tree

class Solution:

# @param num, a list of integers

# @return a tree node

def sortedArrayToBST(self, num):

return self.sortedArray(num,0,len(num) - 1)

def sortedArray(self, num, start, end):

if start > end:

return None

mid = int( (start + end)/2 )

root = TreeNode( num[mid] )

root.left=self.sortedArray(num,start, mid-1)

root.right=self.sortedArray(num,mid+1,end)

return root

Maximum Depth of a Tree

class Solution:

# @param root, a tree node

# @return an integer

def maxDepth(self, root):

if root == None: return 0

left = self.maxDepth(root.left)

right = self.maxDepth(root.right)

if left > right:

left += 1

return left

else:

right += 1

return right

Binary Tree BFS (level order)—O(N)

class Solution:

# @param root, a tree node

# @return a list of lists of integers

def levelOrder(self, root):

if root == None: return []

result = []

thisLevel = [root]

result.append([root.val])

while thisLevel:

nextLevel = []

for i in thisLevel:

if i.left: nextLevel.append(i.left)

if i.right: nextLevel.append(i.right)

if nextLevel:

val = self.storeVal(nextLevel)

result.append(val)

thisLevel = nextLevel

return result

def storeVal(self, arr):

val = []

for i in arr:

val.append(i.val)

return val

Balanced Binary Tree

class Solution:

# @param root, a tree node

# @return a boolean

def isBalanced(self, root):

if root == None: return True

else:

maxHeight = self.isBalance(root)

if maxHeight != False:

return True

else: return False

def isBalance(self, root):

if root == None:

return 0

left = self.isBalance(root.left)

if left is False: return False

right = self.isBalance(root.right)

if right is False: return False

if abs(left - right) > 1: return False

else:

return max(left, right) + 1

Search in Rotated Sorted Array

class Solution:

# @param A, a list of integers

# @param target, an integer to be searched

# @return an integer

def search(self, A, target):

return self.searchHalf(A,0, len(A)-1, target)

def searchHalf(self, A, start, end, target):

if start > end:

return -1

mid = int ((start + end)/ 2)

midVal = A[mid]

startVal = A[start]

endVal = A[end]

if midVal == target:

return mid

# left half is ordered

if midVal >= startVal:

if target<=midVal and target>=startVal:

# target is in left half

return self.searchHalf(A,start,mid-1,target)

else:

return self.searchHalf(A, mid+1, end, target)

# target is in right half

# right half is ordered

elif midVal < startVal:

# target in right half:

if target >= midVal and target <= endVal:

return self.searchHalf(A, mid+1, end, target)

# target in left half:

else:

return self.searchHalf(A, start, mid-1, target)

return -1