'''

Problem:

A binary tree topology is defined as any configuration of the tree,

given a number of nodes, that is agnostic of node labels.

Write an algorithmic function that tells me how many binary tree topologies are possible,

given n nodes in the tree.

Requires:

(1) achieve the algorithm in recursion style (not efficient)

(2) efficient with extra space, record the topology number for each node number

'''

### solution 1 ###

# def binary\_tree\_topo(n):

# if n == 0:

# return 1

# else:

# number = 0

# for i in range(n):

# number += binary\_tree\_topo(i) \* binary\_tree\_topo(n-1-i)

# return number

#

# node\_num = 5

# tree\_topo\_num = binary\_tree\_topo(node\_num)

# print('tree topology number is %d' % tree\_topo\_num)

### solution 2 ###

topo\_doc = [1, 1]

def binary\_tree\_topo(n):

if n == 0 or n == 1:

number = topo\_doc[n]

return number

else:

# notice here, for node n, the value is topo\_doc[n]

for j in range(2, n+1):

node\_add = 0

for i in range(j):

node\_add += topo\_doc[i] \* topo\_doc[j-1-i]

topo\_doc.append(node\_add)

return topo\_doc[n]

number = binary\_tree\_topo(5)

print(number)