131. Palindrome Partitioning

class Solution:

def partition(self, s: str) -> List[List[str]]:

res = []

self.dfs(s, [], res)

return res

def dfs(self, s, path, res):

# 出口，找完将结果加到res里

if not s:

res.append(path)

# 从s的前1、2、3…len(s)个字符串开始判断是否是回文串

for i in range(1, len(s)+1):

if self.isPal(s[:i]):

# 找到回文串后将s的剩余部分再扔到dfs里继续找回文串

self.dfs(s[i:], path+[s[:i]], res)

def isPal(self, s):

return s == s[::-1]

134. Gass Station

class Solution:

def canCompleteCircuit(self, gas: List[int], cost: List[int]) -> int:

if sum(gas) < sum(cost):

return -1

start\_station = 0

current\_tank = 0

for i in range(len(gas)):

current\_tank += gas[i] - cost[i]

if current\_tank < 0:

start\_station = i+1

current\_tank = 0

return start\_station

138. Copy list with random pointer

"""

# Definition for a Node.

class Node:

def \_\_init\_\_(self, val, next, random):

self.val = val

self.next = next

self.random = random

"""

class Solution:

def \_\_init\_\_(self):

self.visited\_dic = {}

def copyRandomList(self, head: 'Node') -> 'Node':

if head == None:

return None

if head in self.visited\_dic:

return self.visited\_dic[head]

node = Node(head.val, None, None)

self.visited\_dic[head] = node

node.next = self.copyRandomList(head.next)

node.random = self.copyRandomList(head.random)

return node