class Solution:

def isPalindrome(self,head):

if head is None:

return True

"""

:type head: ListNode

:rtype: bool

"""

global flag

flag = True

global count

count=1

front = head

def countt(front):

global count

while front.next is not None:

front=front.next

count += 1

global prev

prev = None

countt(head)

if count==1:

return True

#print(count)

front = head

if count % 2 != 0 :

for i in range(count // 2 - 1):

front = front.next

front.next = front.next.next

front = head

for i in range(count // 2 ): # Reverse list

# #print (prev)

# #print(" data ",front.val,end='--> ')

# if prev is not None: #print(prev.val, " is this prev ")

next = front.next

front.next = prev

# if front.next is not None: #print(front.next.val, " is this prev ")

prev = front

front = next

# #print(front.val,front.next. val , end=" ||")

# #print(front.val," after in the middle ")

newfront = front

# #print(newfront.val,"xxxxxxxxxxxxxxxxxxxxxx")

front=prev

# for i in range(count // 2 ):

#

# #print(front.val,".....",i)

# front=front.next

# for i in range(count // 2 ):

#

# #print(newfront.val,".....",i)

# newfront=newfront.next

for i in range(count // 2 ):

global flag

# #print(newfront.val,front.val)

if newfront.val != front.val:

flag = False

front=front.next

newfront=newfront.next

return flag

class Solution:

def isPalindrome(self, head):

"""

:type head: ListNode

:rtype: bool

"""

def get\_length(head):

if head == None:

return 0

length = 1

current = head

while current.next:

length += 1

current = current.next

return length

# recursive function

def isPalindromeUtil(node, length):

if length == 0:

return node, True

if length == 1:

return node.next, True

node\_opposite, is\_palindrome = isPalindromeUtil(node.next, length - 2)

if node.val == node\_opposite.val and is\_palindrome:

return node\_opposite.next, True

else:

return node\_opposite.next, False

length = get\_length(head)

\_, is\_palindrome = isPalindromeUtil(head, length)

return is\_palindrome

class Solution(object):

def isPalindrome(self, head):

"""

:type head: ListNode

:rtype: bool

"""

temp\_head = head

# get length of the linked list

i = 0

while head:

head = head.next

i += 1

ll = i

# reverse the first half list

i = 0

p = temp\_head

reverse = ListNode(None)

while i < ll / 2:

reverse.next, p.next, p, i = p, reverse.next, p.next, i + 1

# if length of the list is odd, then don't compare the middle node

if ll % 2 == 1:

p = p.next

# compare first reversed half list and end halp list

q = reverse.next

while p:

if p.val != q.val:

return False

else:

p, q = p.next, q.next

return True