栈实现队列

class MyQueue:

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

"""

Initialize your data structure here.

"""

self.stack=collections.deque()

self.stack\_tmp=collections.deque()

def push(self, x: int) -> None:

"""

Push element x to the back of queue.

"""

for \_ in range(len(self.stack)):

self.stack\_tmp.appendleft(self.stack.popleft())

self.stack.appendleft(x)

for \_ in range(len(self.stack\_tmp)):

self.stack.appendleft(self.stack\_tmp.popleft())

def pop(self) -> int:

"""

Removes the element from in front of queue and returns that element.

"""

return self.stack.popleft()

def peek(self) -> int:

"""

Get the front element.

"""

return self.stack[0]

def empty(self) -> bool:

"""

Returns whether the queue is empty.

"""

return not self.stack

2的幂：

class Solution:

def isPowerOfTwo(self, n: int) -> bool:

if n==0:

return False

while n%2==0:

n=n//2

return n==1

翻转二叉树

class Solution:

def invertTree(self, root: TreeNode) -> TreeNode:

if not root:

return None

root.left,root.right=root.right,root.left

root.left=self.invertTree(root.left)

root.right=self.invertTree(root.right)

return root

队列实现栈：

class MyStack:

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

"""

Initialize your data structure here.

"""

self.stack=collections.deque()

def push(self, x: int) -> None:

"""

Push element x onto stack.

"""

self.stack.append(x)

for \_ in range(len(self.stack)-1):

self.stack.append(self.stack.popleft())

def pop(self) -> int:

"""

Removes the element on top of the stack and returns that element.

"""

return self.stack.popleft()

def top(self) -> int:

"""

Get the top element.

"""

return self.stack[0]

def empty(self) -> bool:

"""

Returns whether the stack is empty.

"""

return not len(self.stack)

反转链表：

def reverseList(self, head: ListNode) -> ListNode:

new\_head = None

while head :

tmp = head.next # tmp这个变量名指向了head.next（也就是保存了head.next的地址）

head.next = new\_head #head节点的下一个结点变为new\_head

new\_head = head #new\_head这个变量名指向了head所指向（也可以说是保存，便于理解）的东西

head = tmp #head这个变量名指向了tmp（tmp在等号右边，所以和等号左边的tmp不一样 这里是实例化的tmp所指向的东西）

return new\_head

例： a=1 b=2

a=b

这里右边的b就是实例化的2，表示a变量名指向了这个2，当然b也还是指向这个2的

回文链表：

class Solution:

def isPalindrome(self, head: ListNode) -> bool:

slow=head

fast=head

while fast and fast.next:

fast=fast.next.next

slow=slow.next

if fast:

slow=slow.next

#取中点

pre=None

cur=slow

while cur:

tmp=cur.next

cur.next=pre

pre=cur

cur=tmp

#调换

while pre and head:

if pre.val!=head.val:

return False

pre=pre.next

head=head.next

return True

#判段

二叉搜索树的最近公共祖先

def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q: 'TreeNode') -> 'TreeNode':

if (root.val-p.val)\*(root.val-q.val)<=0:

return root

if p.val<root.val and q.val<root.val:

return self.lowestCommonAncestor(root.left,p,q)

if p.val>root.val and q.val>root.val:

return self.lowestCommonAncestor(root.right,p,q)

删除链表中的节点：

class Solution:

def deleteNode(self, node):

"""

:type node: ListNode

:rtype: void Do not return anything, modify node in-place instead.

"""

node.val=node.next.val

node.next=node.next.next

有效的字母异度位词

class Solution:

def isAnagram(self, s: str, t: str) -> bool:

dic\_s={}

for i in s:

if i not in dic\_s:

dic\_s[i]=1

else:

dic\_s[i]+=1

for j in t:

if j not in dic\_s:

return False

else:

dic\_s[j]-=1

for item in dic\_s:

if dic\_s[item]!=0:

return False

return True

各位相加

def addDigits(self, num: int) -> int:

num=str(num)

while len(num)>1:

tmp=0

for i in num:

tmp=tmp+int(i)

num=str(tmp)

return int(num)

方法2

def addDigits(self, num: int) -> int:

while len(str(num))>1:

num=eval("+".join(str(num)))

return num

丑数：

def isUgly(self, num: int) -> bool:

if num==0:

return False

while num%2==0:

num/=2

while num%3==0:

num/=3

while num%5==0:

num/=5

return num==1

缺少数字

def missingNumber(nums):

return (len(nums)+1)\*(len(nums))//2-sum(nums)

第一个错误版本

class Solution:

def firstBadVersion(self, n):

"""

:type n: int

:rtype: int

"""

left=1

right=n

while left<right:

mid=left+(right-left)//2

if isBadVersion(mid)==False:

left=mid+1

else:

right=mid

return left

移动零:

def moveZeroes(self, nums: List[int]) -> None:

"""

Do not return anything, modify nums in-place instead.

"""

for i in nums:

if i==0:

nums.remove(i)

nums.append(i)

return nums

def moveZeroes(nums):

"""

Do not return anything, modify nums in-place instead.

"""

i = j = 0

for i in range(len(nums)):

if nums[i] != 0:

nums[j] , nums[i]= nums[i] , nums[j]

j += 1

[单词规律](https://leetcode-cn.com/problems/word-pattern/)

字典方法：

class Solution:

def wordPattern(self, pattern: str, str: str) -> bool:

str\_list=str.split(" ")

if len(pattern)!=len(str\_list):

return False

str\_dic={}

j=0

for i in str\_list:

if i not in str\_dic:

if pattern[j] in str\_dic.values():

return False

#为了ac [“abbc”,"dog cat cat fish"]的case

str\_dic[i]=pattern[j]

j+=1

elif str\_dic[i]!=pattern[j]:

return False

else:

j+=1

return True

（提取字典的values作为可迭代对象用 dic.values()）

[猜数字游戏](https://leetcode-cn.com/problems/bulls-and-cows/)

def getHint(self, secret: str, guess: str) -> str:

from collections import Counter

countA=0

countB=0

s1=[]

s2=[]

for i in range(len(secret)):

if secret[i]==guess[i]:

countA+=1

else:

s1.append(secret[i])

s2.append(guess[i])

c\_s1=Counter(s1)

c\_s2=Counter(s2)

for k in c\_s1:

if k in c\_s2:

countB+=min(c\_s1[k],c\_s2[k])

return "{}A{}B".format(countA,countB)

“””counter类还是挺方便的”””