1.合并两个排序的整数数组A和B变成一个新的数组。

您在真实的面试中是否遇到过这个题？

Yes

**样例**

给出A=**[1,2,3,4]**，B=**[2,4,5,6]**，返回**[1,2,2,3,4,4,5,6]**

**思路：两幅扑克牌比较大小策略，但是存在一个问题是有一个数组比较完了，另一个数组还有剩余。**

class Solution:

#@param A and B: sorted integer array A and B.

#@return: A new sorted integer array

def mergeSortedArray(self, A, B):

# write your code here

m=len(A)

n=len(B)

A.append(8888)

B.append(8888)

c=[]

for i in range(m+n):

c.append(0)

i=0;j=0

for k in range(m+n):

if A[i] < B[j]:

c[k]=A[i]

i+=1

else:

c[k]=B[j]

j+=1

return c

2. 将两个排序链表合并为一个新的排序链表

您在真实的面试中是否遇到过这个题？

Yes

**样例**

给出 1->3->8->11->15->null，2->null， 返回 1->2->3->8->11->15->null。

思路还是像扑克牌一样一个一个的比较，关键就是谁先到达null的节点。

"""

Definition of ListNode

class ListNode(object):

def \_\_init\_\_(self, val, next=None):

self.val = val

self.next = next

"""

class Solution:

"""

@param two ListNodes

@return a ListNode

"""

def mergeTwoLists(self, l1, l2):

# write your code here

l3=ListNode(0)

p=l3

while l1!=None and l2!=None:

if l1.val<l2.val :

p.next=l1

p=p.next

l1=l1.next

else:

p.next=l2

p=p.next

l2=l2.next

if l1!=None:

while l1 != None:

p.next=l1

p=p.next

l1=l1.next

if l2!=None:

while l2 !=None:

p.next=l2

p=p.next

l2=l2.next

return l3.next

3. 给定一个二叉树，找出其最小深度。

二叉树的最小深度为根节点到最近叶子节点的距离。

您在真实的面试中是否遇到过这个题？

Yes

**样例**

给出一棵如下的二叉树:

        1

     /     \

   2       3

          /    \

        4      5

这个二叉树的最小深度为 2

思路：怎么找到最小深度，就是每个左右节点最小的。如果某个节点左或右不存在那么该节点左右深度就要设为无限大。

class Solution:

"""

@param root: The root of binary tree.

@return: An integer

"""

def minDepth(self, root):

# write your code here

t=mindepth(root,m=0,n=0)

return t

def mindepth(root,m,n):

if root==None:

return 0

if root.left==None and root.right==None:

return 1

m=mindepth(root.left,m,n)

n=mindepth(root.right,m,n)

if m==0:

m=9999

if n==0:

n=9999

return min(m,n)+1

4. 给定一个只含非负整数的m\*n网格，找到一条从左上角到右下角的可以使数字和最小的路径。

你在同一时间只能向下或者向右移动一步

思路：深度优先搜索，或者是动态规划。

深度搜索超时，只能考虑动态规划了。那么到达i，j的最短距离是min（a【i-1】【j】，a[i][j-1]）+nums[i][j]

class Solution:

"""

@param grid: a list of lists of integers.

@return: An integer, minimizes the sum of all numbers along its path

"""

def minPathSum(self, grid):

# write your code here

#cost 309ms

'''

row=len(grid)

colum=len(grid[0])

t=[[0 for a in range(colum+1)]for b in range(row)]

for i in range(row):

t[i][1]=t[i-1][1]+grid[i][0]

for j in range(colum):

t[0][j+1]=t[0][j]+grid[0][j]

for m in range(1,row):

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

t[m][n]=min(t[m-1][n],t[m][n-1])+grid[m][n-1]

return t[-1][-1]

'''

row=len(grid)

colum=len(grid[0])

temp=[8888]

self.srarch(grid,0,0,0,row,colum,temp)

return temp[-1]

def srarch(self,a,i,j,sum1,row,colum,temp):

sum1+=a[i][j]

if i==row-1 and j==colum-1:

if sum1<temp[-1]:

temp[-1]=sum1

if i+1<row:

self.srarch(a,i+1,j,sum1,row,colum,temp)

if j+1<colum:

self.srarch(a,i,j+1,sum1,row,colum,temp)

5. 给定一个整数数组，找到一个具有最小和的子数组。返回其最小和。

 注意事项

子数组最少包含一个数字

您在真实的面试中是否遇到过这个题？

Yes

**样例**

给出数组**[1, -1, -2, 1]**，返回 -3

思路：动态规划吧，二维数组表示和，每次寻找最小的，这样每次最小值都会有三个候选，a[i][j]=min(a[i][j-1],a[i][j-1]+nums[j],nums[j])或者要重小于零开始，如果没有小于零就选择最小值。做个最小值，然后大于零时从头开始，这样复杂度在o（n）

class Solution:

"""

@param nums: a list of integers

@return: A integer denote the sum of minimum subarray

"""

def minSubArray(self, nums):

# write your code here

'''第一种方法算法复杂度太高

a=[0]

length=len(nums)+1

for i in range(1,length):

a.append(a[i-1]+nums[i-1])

min=8888

for j in range(1,len(a)):

for h in range(j):

temp=a[j]-a[h]

if temp<min:

min=temp

return min

'''

#第二中方法用时563ms

a=[]

length=len(nums)

sum=0

min=8888

for i in range(0,length):

sum+=nums[i]

if sum<min:

min=sum

if sum>0:

sum=0

return min

6. 给一个数组 *nums* 写一个函数将 0 移动到数组的最后面，非零元素保持原数组的顺序

 注意事项

1.必须在原数组上操作  
2.最小化操作数

您在真实的面试中是否遇到过这个题？

Yes

**样例**

给出 *nums* = [0, 1, 0, 3, 12], 调用函数之后, *nums* = [1, 3, 12, 0, 0].

思路：是零就删除，记录下个数到最后在补下。不然一个一个交换，碰到零和一起交换，麻烦死了。

class Solution:

# @param {int[]} nums an integer array

# @return nothing, do this in-place

def moveZeroes(self, nums):

# Write your code here

i=0

length=len(nums)

j=0

while j< length:

if nums[i]==0:

del(nums[i])

j+=1

nums.append(0)

else:

i+=1

j+=1

return nums

7. 找到单链表倒数第n个节点，保证链表中节点的最少数量为n。

您在真实的面试中是否遇到过这个题？

Yes

**样例**

给出链表**3->2->1->5->null**和n = 2，返回倒数第二个节点的值1.

思路把链表换成list然后就可以直接按序号访问了。

"""

Definition of ListNode

class ListNode(object):

def \_\_init\_\_(self, val, next=None):

self.val = val

self.next = next

"""

class Solution:

"""

@param head: The first node of linked list.

@param n: An integer.

@return: Nth to last node of a singly linked list.

"""

def nthToLast(self, head, n):

# write your code here

a=[]

if head==None:

return None

while head!=None:

a.append(head)

head=head.next

return a[-n]

8. 给一个01矩阵，求不同的岛屿的个数。

0代表海，1代表岛，如果两个1相邻，那么这两个1属于同一个岛。我们只考虑上下左右为相邻。

您在真实的面试中是否遇到过这个题？

Yes

**样例**

在矩阵：

[

[1, 1, 0, 0, 0],

[0, 1, 0, 0, 1],

[0, 0, 0, 1, 1],

[0, 0, 0, 0, 0],

[0, 0, 0, 0, 1]

]

中有 3 个岛.

思路我要对1做上下左右深度搜索，边界条件，访问过记录，返回什么，如果找到1，返回1并标记为访问过，剩下的就是遍历整个矩阵了。

class Solution:

# @param {boolean[][]} grid a boolean 2D matrix

# @return {int} an integer

def numIslands(self, grid):

# Write your code here

row=len(grid)

if row==0:

return 0

colum=len(grid[0])

b = [[1 for m in range(colum)] for k in range(row)]

num=0

for i in range(row):

#print(i)

for j in range(colum):

if grid[i][j] and b[i][j]:

dfs(grid,b,i,j,row,colum)

num+=1

return num

def dfs(grid,b,i,j,row,colum):

if i<0 or i>=row:

return 1

if j<0 or j>=colum :

return 1

if (not grid[i][j]) or not (b[i][j]):

return 1

b[i][j]=0

dfs(grid,b,i-1,j,row,colum)

dfs(grid,b,i+1,j,row,colum)

dfs(grid,b,i,j-1,row,colum)

dfs(grid,b,i,j+1,row,colum)

9. 用 O(*1*) 时间检测整数 *n* 是否是 *2* 的幂次。

您在真实的面试中是否遇到过这个题？

Yes

**样例**

n=4，返回 true;

n=5，返回 false.

思路：不知道看答案吧。

class Solution:

"""

@param n: An integer

@return: True or false

"""

def checkPowerOf2(self, n):

# write your code here

if n>0 and n&(n-1)==0:

return True

return False

10. 我们有一个栅栏，它有n个柱子，现在要给柱子染色，有k种颜色可以染。  
必须保证不存在超过2个相邻的柱子颜色相同，求有多少种染色方案。

 注意事项

n和k都是非负整数

您在真实的面试中是否遇到过这个题？

Yes

**样例**

n = 3, k = 2, return 6

post 1, post 2, post 3

way1 0 0 1

way2 0 1 0

way3 0 1 1

way4 1 0 0

way5 1 0 1

way6 1 1 0

思路：很显然是动态规划：当相同颜色就是an-2，当不同颜色就是an-1。（因为理论上最后一个柱子可以取到任何颜色，那么最后\*k-1）

初始值应该是k和k平方，

class Solution:

# @param {int} n non-negative integer, n posts

# @param {int} k non-negative integer, k colors

# @return {int} an integer, the total number of ways

def numWays(self, n, k):

# Write your code here

if k==1 and n>2:

return 0

if 1==n:

return k

if 2==n:

return k\*\*2

return (self.numWays(n-1, k)+self.numWays(n-2, k))\*(k-1)