【例1】反转一个3位整数

3.代码实现

class Solution:

#参数number: 一个三位整数

#返回值: 反转后的数字

def reverseInteger(self, number):

h = int(number/100)

t = int(number%100/10)

z = int(number%10)

return (100\*z+10\*t+h)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

num = 123

ans = solution.reverseInteger(num)

print("输入：", num)

print("输出：", ans)

【例2】合并排序数组

3.代码实现

class Solution:

#参数A: 有序整数数组A

#参数B: 有序整数数组B

#返回:一个新的有序整数数组

def mergeSortedArray(self, A, B):

i, j = 0, 0

C = []

while i < len(A) and j < len(B):

if A[i] < B[j]:

C.append(A[i])

i += 1

else:

C.append(B[j])

j += 1

while i < len(A):

C.append(A[i])

i += 1

while j < len(B):

C.append(B[j])

j += 1

return C

#主函数

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

A = [1,4]

B = [1,2,3]

D = [1,2,3,4]

E = [2,4,5,6]

solution = Solution()

print("输入：", A, " ", B)

print("输出：", solution.mergeSortedArray(A,B))

print("输入：", D, " ", E)

print("输出：", solution.mergeSortedArray(D,E))

【例3】旋转字符串

3.代码实现

class Solution:

#参数s:字符列表

#参数offset:整数

#返回值:无

def rotateString(self, s, offset):

if len(s) > 0:

offset = offset % len(s)

temp = (s + s)[len(s) - offset : 2 \* len(s) - offset]

for i in range(len(temp)):

s[i] = temp[i]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = ["a","b","c","d","e","f","g"]

offset = 3

solution = Solution()

solution.rotateString(s, offset)

print("输入：s =", ["a","b","c","d","e","f","g"], " ", "offset =",offset)

print("输出：s =", s)

【例4】相对排名

3.代码实现

class Solution:

#参数nums为整数列表

#返回列表

def findRelativeRanks(self, nums):

score = {}

for i in range(len(nums)):

score[nums[i]] = i

sortedScore = sorted(nums, reverse=True)

answer = [0] \* len(nums)

for i in range(len(sortedScore)):

res = str(i + 1)

if i == 0:

res = 'Gold Medal'

if i == 1:

res = 'Silver Medal'

if i == 2:

res = 'Bronze Medal'

answer[score[sortedScore[i]]] = res

return answer

#主函数

if \_\_name\_\_ == '\_\_main\_\_':  
 num = [5,4,3,2,1]  
 s = Solution()  
 print("输入为：",num)

print("输出为：",s.findRelativeRanks(num))

【例5】二分查找

3.代码实现

class Solution:

#参数nums: 整数数组

#参数target: 要查找的目标数字

#返回值：目标数字的第一个位置，从0开始

def binarySearch(self, nums, target):

return self.search(nums, 0, len(nums) - 1, target)

def search(self, nums, start, end, target):

if start > end:

return -1

mid = (start + end)//2

if nums[mid] > target:

return self.search(nums, start, mid, target)

if nums[mid] == target:

return mid

if nums[mid] < target:

return self.search(nums, mid, end, target)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

my\_solution = Solution()

nums = [1,2,3,4,5,6]

target = 3

targetIndex = my\_solution.binarySearch(nums, target)

print("输入：nums =", nums, " ", "target =",target)

print("输出：",targetIndex)

【例6】下一个更大的数

3.代码实现

class Solution:

#参数nums1为整数数组

#参数nums2为整数数组

#返回整数数组

def nextGreaterElement(self, nums1, nums2):

answer = {}

stack = []

for x in nums2:

while stack and stack[-1] < x:

answer[stack[-1]] = x

del stack[-1]

stack.append(x)

for x in stack:

answer[x] = -1

return [answer[x] for x in nums1]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()  
 nums1 = [4,1,2]  
 nums2 = [1,3,4,2]  
 print("输入1为：",nums1)

print("输入2为：",nums2)

print("输出为 ：",s.nextGreaterElement(nums1,nums2))

【例7】字符串中的单词数

3.代码实现

class Solution:

#参数s为字符串

#返回整数

def countSegments(self, s):

res = 0

for i in range(len(s)):

if s[i] != ' ' and (i == 0 or s[i - 1] == ' '):

res += 1

return res

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n = "Hello, my name is John"

print("输入为：",n)

print("输出为：",s.countSegments(n))

【例8】勒索信

3.代码实现

class Solution:

"""

参数ransomNote为字符串

参数magazine为字符串

返回布尔类型

"""

def canConstruct(self, ransomNote, magazine):

arr = [0] \* 26

for c in magazine:

arr[ord(c) - ord('a')] += 1

for c in ransomNote:

arr[ord(c) - ord('a')] -= 1

if arr[ord(c) - ord('a')] < 0:

return False

return True

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

ransomNote = "aa"

magazine = "aab"

print("输入勒索信为：",ransomNote)

print("输入杂志内容：",magazine)

print("输出为：",s.canConstruct(ransomNote,magazine))

【例9】不重复的两个数

3.代码实现

#参数arr是输入的待查数组

#返回值是两个值的列表，内容没有重复的

class Solution:

def theTwoNumbers(self, a):

ans = [0, 0]

for i in a:

ans[0] = ans[0] ^ i

c = 1

while c & ans[0] != c:

c = c << 1

for i in a:

if i & c == c:

ans[1] = ans[1] ^ i

ans[0] = ans[0] ^ ans[1]

return ans

if \_\_name\_\_ == '\_\_main\_\_':

arr = [1, 2, 5, 1]

solution = Solution()

print(" 数组为：", arr)

print(" 两个没有重复的数字是:", solution.theTwoNumbers(arr))

【例10】双胞胎字符串

3.代码实现

#参数s和t是一对字符串

#返回值是个字符串，能否根据规则转换

class Solution:

def isTwin(self, s, t):

if len(s) != len(t):

return "No"

oddS = []

evenS = []

oddT = []

evenT = []

for i in range(len(s)):

if i & 1:

oddS.append(s[i])

oddT.append(t[i])

else :

evenS.append(s[i])

evenT.append(t[i])

oddS.sort()

oddT.sort()

evenS.sort()

evenT.sort()

for i in range (len(oddS)) :

if oddS[i] != oddT[i]:

return "No"

for i in range (len(evenS)) :

if evenS[i] != evenT[i]:

return "No"

return "Yes"

if \_\_name\_\_ == '\_\_main\_\_':

s="abcd"

t="cdab"

solution = Solution()

print(" s与t分别为：", s, t)

print(" 是否:", solution.isTwin(s, t))

【例11】最接近target的值

3.代码实现

#参数array是输入列表

#参数target是目标值

#返回值是整数

class Solution:

def closestTargetValue(self, target, array):

n = len(array)

if n < 2:

return -1

array.sort()

diff = 0x7fffffff

left = 0

right = n - 1

while left < right:

if array[left] + array[right] > target:

right -= 1

else:

diff = min(diff, target - array[left] - array[right])

left += 1

if diff == 0x7fffffff:

return -1

else:

return target - diff

if \_\_name\_\_ == '\_\_main\_\_':

array = [1,3,5,11,7]

target = 15

solution = Solution()

print(" 输入数组为：", array,"目标值为：", target)

print(" 最近可以得到值为:", solution.closestTargetValue(target, array))

【例12】点积

3.代码实现

#参数A和B是输入列表

#返回值是个整数，是点积

class Solution:

def dotProduct(self, A, B):

if len(A) == 0 or len(B) == 0 or len(A) != len(B):

return -1

ans = 0

for i in range(len(A)):

ans += A[i] \* B[i]

return ans

if \_\_name\_\_ == '\_\_main\_\_':

A = [1,1,1]

B = [2,2,2]

solution = Solution()

print(" A与B分别为：", A, B)

print(" 点积为：", solution.dotProduct(A, B))

【例13】函数运行时间

3.代码实现

#参数s为输入原始字符串

#返回值是个字符串，意为每个名字的函数运行了多久

class Solution:

def getRuntime(self, a):

map={}

for i in a:

count = 0

while not i[count] == ' ':

count = count + 1

fun = i[0 : count]

if i[count+2] == 'n':

count = count + 7

v = int(i[count:len(i)])

if fun in map.keys():

map[fun] = v - map[fun]

else:

map[fun] = v

else:

count = count + 6

v = int(i[count:len(i)])

map[fun] = v - map[fun]

res=[]

for i in map:

res.append(i)

res.sort()

for i in range(0,len(res)):

res[i] = res[i] + '|' + str(map[res[i]])

return res

if \_\_name\_\_ == '\_\_main\_\_':

s = ["F1 Enter 10","F2 Enter 18","F2 Exit 19","F1 Exit 20"]

solution = Solution()

print(" 输入运行时间为：", s)

print(" 每个输出时间为:", solution.getRuntime(s))

【例14】查询区间

1.问题描述

给定一个包含若干个区间的List数组，长度是1000，例如，[500,1500]，[2100,3100].给定一个number，number是否在这些区间内，返回True或False。

2.问题示例

输入是 List = [[100,1100],[1000,2000],[5500,6500]]和number = 6000，输出是True，6000在区间[5500,6500]。输是List = [[100,1100],[2000,3000]]和number = 3500，输出是 False，3500不在list的任何一个区间中。

3.代码实现

#参数List是区间列表

#参数number是待查数字

#返回值是个字符串，True或者False

class Solution:

def isInterval(self, intervalList, number):

high = len(intervalList) - 1

low = 0

while high >= low:

if 0 < (number - intervalList[(high + low)//2][0]) <= 1000:

return 'True'

elif 1000 < number - intervalList[(high + low)//2][0]:

low = (high + low) // 2 + 1

elif 0 > number - intervalList[(high + low)//2][0]:

high = (high + low) // 2 - 1

return 'False'

if \_\_name\_\_ == '\_\_main\_\_':

number = 6000

intervalList = [[100,1100],[1000,2000],[5500,6500]]

solution = Solution()

print(" 区间List为：", intervalList)

print(" 数字为：", number)

print(" 是否在区间中:", solution.isInterval(intervalList, number))

4.运行结果

输入区间List为： [[100, 1100], [1000, 2000], [5500, 6500]]

输出数字为： 6000

是否在区间中: True

【例15】飞行棋

3.代码实现

#参数length是棋盘长度（不包含起始点）

#参数connections是跳点的集合

#返回值是个整数，代表最小步数

class Solution:

def modernLudo(self, length, connections):

ans = [i for i in range(length+1)]

for i in range(length+1):

for j in range(1,7):

if i - j >= 0:

ans[i] = min(ans[i], ans[i-j]+1)

for j in connections:

if i == j[1]:

ans[i] = min(ans[i], ans[j[0]])

return ans[length]

#SPFA解法

class Solution:

"""

参数length为棋盘长度

参数connections为连接位置表

返回最小次数

"""

def modernLudo(self, length, connections):

dist = [1000000000 for i in range(100050)]

vis = [0 for i in range(100050)]

Q = [0 for i in range(100050)]

st = 0

ed = 0

dist[1] =0

vis[1] = 1

Q[ed] = 1;

ed += 1

while(st<ed) :

u = Q[st]

st += 1

vis[u] = 0

for roads in connections :

if(roads[0] != u):

continue

v = roads[1]

if(dist[v] > dist[u]):

dist[v] = dist[u]

if(vis[v] == 0) :

vis[v] = 1

Q[ed] = v

ed += 1

for i in range(1, 7):

if (i + u > length):

break

v = i + u

if(dist[v] > dist[u] + 1) :

dist[v] = dist[u] + 1

if(vis[v] == 0):

vis[v] = 1

Q[ed] = v

ed += 1

return dist[length]

if \_\_name\_\_ == '\_\_main\_\_':

length = 15

connections = [[2, 8],[6, 9]]

solution = Solution()

print(" 棋盘长度为：", length)

print(" 连接为：", connections)

print(" 最小需要为:", solution.modernLudo(length, connections))

【例16】移动石子

3.代码实现

#参数arr是一个列表

#返回值为整数，为最小移动次数

class Solution:

def movingStones(self, arr):

arr = sorted(arr)

even = 0

odd = 0

for i in range(0,len(arr)):

odd += abs(arr[i]-(2\*i+1))

even += abs(arr[i] - (2\*i+2))

if odd < even:

return odd

return even

if \_\_name\_\_ == '\_\_main\_\_':

arr = [1, 6, 7, 8, 9]

solution = Solution()

print(" 数组是：", arr)

print(" 最小移动数是:", solution.movingStones(arr))

【例17】数组剔除元素后的乘积

3.代码实现

class Solution:

#参数A: 整数数组A

#返回值: 整数数组B

def productExcludeItself(self, A):

length ,B = len(A) ,[]

f = [ 0 for i in range(length + 1)]

f[ length ] = 1

for i in range(length - 1 , 0 , -1):

f[ i ] = f[ i + 1 ] \* A[ i ]

tmp = 1

for i in range(length):

B.append(tmp \* f[ i + 1 ])

tmp \*= A[ i ]

return B

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

A = [1, 2, 3, 4]

B = solution.productExcludeItself(A)

print("输入：", A)

print("输出：", B)

【例18】键盘的一行

3.代码实现

class Solution:

#参数words为字符串列表

#返回字符串列表

def findWords(self, words):

res = []

s = ["qwertyuiop", "asdfghjkl", "zxcvbnm"]

for w in words:

for j in range(3):

flag = 1

for i in w:

if i.lower() not in s[j]:

flag = 0

break

if flag == 1:

res.append(w)

break

return res

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

word = ["Hello", "Alaska", "Dad", "Peace"]  
 s = Solution()  
 print("输入为：",word)

print("输出为：",s.findWords(word))

【例19】第n个数位

3.代码实现

class Solution:

"""

参数n为整数

返回整数

"""

def findNthDigit(self, n):

# 初始化一位数的个数为9，从1开始

length = 1

count = 9

start = 1

while n > length \* count:

# 以此类推二位数的个数为90，从10开始

n -= length \* count

length += 1

count \*= 10

start \*= 10

# 找到第n位数所在的整数start

start += (n - 1) // length

return int(str(start)[(n - 1) % length])

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n = 11

print("输入为：",n)

print("输出为：",s.findNthDigit(n))

【例20】找不同

3.代码实现

class Solution:

"""

参数s为字符串

参数t为字符串

返回字符

"""

def findTheDifference(self, s, t):

flag = 0

for i in range(len(s)):

# 计算每一位字符的Ascll码之差

flag += (ord(t[i]) - ord(s[i]))

flag += ord(t[-1])

return chr(flag)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n = "abcd"

t = "abcde"

print("输入字符串1为：",n)

print("输入字符串2为：",t)

print("输出插入字符为：",s.findTheDifference(n,t))

【例21】第k个组合

3.代码实现

#参数k代表寻找的组数

#参数n代表有多少人

#返回值是一个列表，是目标数组里的按序排列

class Solution:

def getCombination(self, n, k):

C = [[0] \* (n + 1) for \_ in range(n + 1)]

for i in range(n + 1):

C[i][0] = 1

C[i][i] = 1

# Compute C(m, n) using C(m, n) = C(m-1, n-1)+C(m-1, n).

for i in range(1, n + 1):

for j in range(1, i):

C[i][j] = C[i - 1][j - 1] + C[i - 1][j]

ans = []

curr\_index = 1

for i in range(1, n // 2 + 1):

base = C[n - curr\_index][n // 2 - i]

# Search for next digit in ans.

while k > base:

curr\_index = curr\_index + 1

base = base + C[n - curr\_index][n // 2 - i]

base = base - C[n - curr\_index][n // 2 - i]

k = k - base;

ans.append(curr\_index)

curr\_index = curr\_index + 1

return ans

if \_\_name\_\_ == '\_\_main\_\_':

n = 8

k = 11

solution = Solution()

print(" 人数为：", n, " 找第k组：", k)

print(" 第k组为:", solution.getCombination(n, k))

【例22】平面列表

3.代码实现

class Solution(object):

#参数nestedList： 一个列表，列表中的每个元素都可以是一个列表或整数

#返回值：一个整数列表

def flatten(self, nestedList):

stack = [nestedList]

flatten\_list = []

while stack:

top = stack.pop()

if isinstance(top, list):

for elem in reversed(top):

stack.append(elem)

else:

flatten\_list.append(top)

return flatten\_list

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

nums = [[1,2],2,[1,1,3]]

flatten\_list = solution.flatten(nums)

print("输入：", nums)

print("输出：", flatten\_list)

【例23】子域名访问计数

2.问题示例

例如，输入["9001 school.bupt.edu"]，输出["9001 school.bupt.edu", "9001 bupt.edu", "9001 edu"]，只有一个域名："school.bupt.edu"，如题所述，子域名"bupt.edu"和"edu"也会被访问，所以要访问9001次。

3.代码实现

class Solution:

# 利用hash表，对子域名计数。注意对字符串的划分

def subdomainVisits(self, cpdomains):

count = {}

for domain in cpdomains:

visits = int(domain.split()[0])

domain\_segments = domain.split()[1].split('.')

top\_level\_domain = domain\_segments[-1]

sec\_level\_domain = domain\_segments[-2] + '.' + domain\_segments[-1]

count[top\_level\_domain] = count[top\_level\_domain] + visits if top\_level\_domain in count.keys() else visits

count[sec\_level\_domain] = count[sec\_level\_domain] + visits if sec\_level\_domain in count.keys() else visits

if domain.count('.') == 2:

count[domain.split()[1]] = count[domain.split()[1]] + visits if domain.split()[1] in count.keys() else visits

return [str(v) + ' ' + k for k,v in count.items()]

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

inputnum=["1201 school.bupt.edu"]

print("输入为：",inputnum)

print("输入为：",solution.subdomainVisits(inputnum))  
4.运行结果

输入为： ['1201 school.bupt.edu']

输入为： ['1201 edu', '1201 bupt.edu', '1201 school.bupt.edu']

【例24】最长AB子串

3.代码实现

#参数S是待查字符串

#返回值是一个整数，是最大字符串长度

class Solution:

def getAns(self, S):

ans = 0

arr = [0 for i in range(len(S))]

sets = {}

if S[0] == 'A':

arr[0] = 1

sets[1] = 0

else:

arr[0] = -1

sets[-1] = 0

for i in range(1, len(S)):

if S[i] == 'A':

arr[i] = arr[i - 1] + 1

if arr[i] == 0:

ans = i + 1

continue

if arr[i] in sets:

ans = max(ans, i - sets[arr[i]])

else:

sets[arr[i]] = i

else:

arr[i] = arr[i - 1] - 1

if arr[i] == 0:

ans = i + 1

continue

if arr[i] in sets:

ans = max(ans, i - sets[arr[i]])

else:

sets[arr[i]] = i

return ans

if \_\_name\_\_ == '\_\_main\_\_':

S = "ABABAB"

solution = Solution()

print(" AB字符串为：", S)

print(" 最长AB出现次数相同的子字符串长度是:", solution.getAns(S))

【例25】删除字符

3.代码实现

#参数s是待删除字符的原字符串

#参数t是目标字符串

#返回值是一个布尔值，意为能否由s删除一些字符得到t

class Solution:

def canGetString(self, s, t):

pos = 0

for x in t:

while pos < len(s) and s[pos] != x:

pos += 1

if pos == len(s):

return False

pos += 1

return True

if \_\_name\_\_ == '\_\_main\_\_':

s="abc"

t="c"

solution = Solution()

print(" 原string和目标string分别为：", s, t)

print(" 能否实现:", solution.canGetString(s, t))

【例26】字符串写入的行数

3.代码实现

class Solution(object):

def numberOfLines(self, widths, S):

#参数widths为数组

#参数S为字符串

#返回数组

line = 1

space = 0

flag = False

for c in S:

if flag:

line += 1

flag = False

space += widths[ord(c) - 97]

if space > 100:

line += 1

space = widths[ord(c) - 97]

elif space == 100:

space = 0

flag = True

return [line, space]

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

width=[10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10,10]

s="abcdefghijklmnopqrstuvwxyz"

print("输入字符宽度为：",width)

print("输入的字符串为：",s)

print("输出为：",solution.numberOfLines(width,s))

【例27】独特的摩尔斯编码

3.代码实现

class Solution:

def uniqueMorseRepresentations(self, words):

#参数words为列表

#返回整数

# 用set保存出现过的摩斯码即可

morse = [".-","-...","-.-.","-..",".","..-.","--.","....","..",".---","-.-",".-..","--",

"-.","---",".--.","--.-",".-.","...","-","..-","...-",".--","-..-","-.--","--.."]

s = set()

for word in words:

tmp = ''

for w in word:

tmp += morse[ord(w) - 97]

s.add(tmp)

return len(s)

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

inputnum=["gin", "zen", "gig", "msg"]

print("输入为：",inputnum)

print("输出为：",solution.uniqueMorseRepresentations(inputnum))

【例28】比较字符串

3.代码实现

class Solution:

#参数A : 包括大写字母的字符串

#参数B : 包括大写字母的字符串

#返回值: 如果字符串A包含B中的所有字符，返回True，否则返回False

def compareStrings(self, A, B):

if len(B) == 0:

return True

if len(A) == 0:

return False

#trackTable首先记录A中所有的字符以及它们的个数，然后遍历B,如果出现trackTable[i]小于0的情况，说明B中该字符出现的次数大于在A中出现的次数

trackTable = [0 for \_ in range(26)]

for i in A:

trackTable[ord(i) - 65] += 1

for i in B:

if trackTable[ord(i) - 65] == 0:

return False

else:

trackTable[ord(i) -65] -= 1

return True

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

A = "ABCD"

B = "ACD"

print("输入：", A, B)

print("输出：", solution.compareStrings(A,B))

【例29】能否转换

3.代码实现

#参数S和T为原始字符串和目标字符串

#返回值是布尔值，代表能否转换

class Solution:

def canConvert(self, s, t):

j = 0

for i in range(len(s)):

if s[i] == t[j]:

j += 1

if j == len(t):

return True

return False

if \_\_name\_\_ == '\_\_main\_\_':

s = "longterm"

t = "long"

solution = Solution()

print(" S与T分别为：", s, t)

print(" 能否删除得到:", solution.canConvert(s, t))

【例30】经典二分查找问题

3.代码实现

#参数nums是一个整型排序数组

#参数target是一个任意整型数

#返回值是一个整型数，若nums存在，返回该数位置；若不存在，返回-1

class Solution:

def findPosition(self, nums, target):

if len(nums) is 0:

return -1

start = 0

end = len(nums)-1

while start + 1 < end :

mid = start + (end-start)//2

if nums[mid] == target:

end = mid

elif nums[mid] < target:

start = mid

else:

end = mid

if nums[start] == target:

return start

if nums[end] == target:

return end

return -1

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generator=[1,2,2,4,5,5]

target = 2

solution = Solution()

print("输入:", generator)

print("输出:", solution. myAtoi(generator, target))

【例31】抽搐词 难度等级★

3.代码实现

class Solution:

def twitchWords(self, str):

n = len(str)

c = str[0]

left = 0

ans = []

for i in range(n):

if str[i] != c:

if i - left >= 3:

ans.append([left, i - 1])

c = str[i]

left = i

if n - left >= 3:

ans.append([left, n - 1])

return ans

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

str = "whooooisssbesssst"

solution = Solution()

print(" 输入为：", str)

print(" 输出为：", solution.twitchWords(str))

【例32】排序数组中最接近元素

3.代码实现

#参数nums是一个整型排序数组

#参数target是一个整型数

#返回值是这个数组中最接近target的整数

class Solution:

def findPosition(self, A, target):

if not A:

return -1

start, end = 0,len(A)-1

while start+ 1<end:

mid = start +(end-start)//2

if A[mid]<target:

start= mid

elif A[mid]>target:

end = mid

else:

return mid

if target-A[start]<A[end]-target:

return start

else:

return end

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generator = [1,4,6]

target = 3

solution = Solution()

print("输入：", generator,",target =",target)

print("输出：", solution.findPosition(generator, target))

【例33】构造矩形

3.代码实现

class Solution:

#参数area为整数

#返回为整数

def constructRectangle(self, area):

import math

W = math.floor(math.sqrt(area))

while area % W != 0:

W -= 1

return [area // W, W]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

area =4

print("输入面积为：",area)

print("输出长宽为：",s.constructRectangle(area))

【例34】两个排序数组合的第k小元素

3.代码实现

#参数A，B是整型排序数组

#参数k是一个整型数，表示第k小

#返回值是数组中第k小的整数

class Solution:

def kthSmallestSum(self, A, B, k):

if not A or not B:

return None

n, m = len(A), len(B)

minheap = [(A[0] + B[0], 0, 0)]

visited = set([0])

num = None

for \_ in range(k):

num, x, y = heapq.heappop(minheap)

if x + 1 < n and (x + 1) \* m + y not in visited:

heapq.heappush(minheap, (A[x + 1] + B[y], x + 1, y))

visited.add((x + 1) \* m + y)

if y + 1 < m and x \* m + y + 1 not in visited:

heapq.heappush(minheap, (A[x] + B[y + 1], x, y + 1))

visited.add(x \* m + y + 1)

return num

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generator\_A = [1,7,11]

generator\_B = [2,4,6]

k = 3

solution = Solution()

print("输入:", generator\_A,generator\_B)

print("k= ",k)

print("输出:", solution.kthSmallestSum(generator\_A,generator\_B, k))

【例35】玩具工厂

3.代码实现

#参数type是一个字符串，表示不同玩具类型

#返回值是不同类型对应的玩具对象

class Toy:

def talk(self):

raise NotImplementedError('This method should have implemented.')

class Dog(Toy):

def talk(self):

print ("Wow")

class Cat(Toy):

def talk(self):

print ("Meow")

class ToyFactory:

def getToy(self, type):

if type == 'Dog':

return Dog()

elif type == 'Cat':

return Cat()

return None

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

ty = ToyFactory()

type='Dog'

type1='Cat'

toy = ty.getToy(type)

print("输入：type= Dog，输出：")

toy.talk()

toy = ty.getToy(type1)

print("输入：type= Cat，输出：")

toy.talk()

【例36】形状工厂

3.代码实现

#参数shapeType是一个字符串，表示不同形状

#返回值是不同对象，Triangle，Square，Rectangle

class Shape:

def draw(self):

raise NotImplementedError('This method should have implemented.')

class Triangle(Shape):

def draw(self):

print(" /\\")

print(" / \\")

print("/\_\_\_\_\\")

class Rectangle(Shape):

def draw(self):

print(" ----")

print("| |")

print(" ----")

class Square(Shape):

def draw(self):

print( " ----")

print( "| |")

print( "| |")

print( " ----")

class ShapeFactory:

def getShape(self, shapeType):

if shapeType == "Triangle":

return Triangle()

elif shapeType == "Rectangle":

return Rectangle()

elif shapeType == "Square":

return Square()

else:

return None

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

sf = ShapeFactory()

shapeType='Triangle'

shape = sf.getShape(shapeType)

print("输入：type= Triangle，\n输出：")

shape.draw()

shapeType1='Rectangle'

shape = sf.getShape(shapeType1)

print("输入：type= Rectangle，\n输出：")

shape.draw()

shapeType2='Square'

shape = sf.getShape(shapeType2)

print("输入：type= Square，\n输出：")

shape.draw()

【例37】二叉树最长连续序列

3.代码实现

#参数root是一个二叉树的根

#返回值是此二叉树中最长连续序列

class TreeNode:

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

self.val = val

self.left = None

self.right = None

class Solution:

def longestConsecutive(self, root):

return self.helper(root, None, 0)

def helper(self, root, parent, len):

if root is None:

return len

if parent != None and root.val == parent.val + 1:

len += 1

else:

len = 1

return max(len, max(self.helper(root.left, root, len), \

self.helper(root.right, root, len)))

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

root = TreeNode(1)

root.right = TreeNode(3)

root.right.left = TreeNode(2)

root.right.right = TreeNode(4)

root.right.right.right = TreeNode(5)

solution= Solution()

print("输入是： {1,#,3,2,4,#,#,#,5}")

print("输出是：", solution.longestConsecutive(root))

【例38】首字母大写

3.代码实现

class Solution:

#参数s为字符串

#返回字符串

def capitalizesFirst(self, s):

n = len(s)

s1 = list(s)

if s1[0] >= 'a' and s1[0] <= 'z':

s1[0] = chr(ord(s1[0]) - 32)

for i in range(1, n):

if s1[i - 1] == ' ' and s1[i] != ' ':

s1[i] = chr(ord(s1[i]) - 32)

return ''.join(s1)

if \_\_name\_\_ == '\_\_main\_\_':

s = "i am from bupt"

solution = Solution()

print("输入为：",s)

print("输出为：",solution.capitalizesFirst(s))

【例39】7进制

3.代码实现

class Solution:

#参数num为10进制整数

#返回7进制整数

#不断执行对7取模和取整操作，直到商小于7

def convertToBase7(self, num):

if num < 0:

return '-' + self.convertToBase7(-num)

if num < 7:

return str(num)

return self.convertToBase7(num // 7) + str(num % 7)

if \_\_name\_\_ == '\_\_main\_\_':

num = 777

solution = Solution()

print("输入为：",num)

print("输出为：",solution.convertToBase7(num))

【例40】查找数组中没有出现的所有数字

3.代码实现

class Solution:

#参数为nums整数列表

#返回整数列表

def findDisappearedNumbers(self, nums):

n = len(nums)

s = set(nums)

res = [i for i in range(1, n+1) if i not in s]

return res

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n = [4,3,2,7,8,2,3,1]

print("输入为：",n)

print("输出为：",s.findDisappearedNumbers(n))

【例41】回旋镖的数量

3.代码实现

class Solution(object):

def getDistance(self, a, b):

dx = a[0] - b[0]

dy = a[1] - b[1]

return dx \* dx + dy \* dy

def numberOfBoomerangs(self, points):

#参数points为整数列表

#返回整数

if points == None:

return 0

ans = 0

for i in range(len(points)):

disCount = {}

for j in range(len(points)):

if i == j:

continue

distance = self.getDistance(points[i], points[j])

count = disCount.get(distance, 0)

disCount[distance] = count + 1

for distance in disCount:

ans += disCount[distance] \* (disCount[distance] - 1)

return ans

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n = [[0,0],[1,0],[2,0]]

print("输入为：",n)

print("输出为：",s.numberOfBoomerangs(n))

【例42】合并排序数组

3.代码实现

class Solution:

#参数A: 已排序整数数组A有m个元素，但是A的大小是m+n

#参数m: 整数

#参数B: 已排序整数数组B，它有n个元素

#参数n: 整数

#返回值: 无

def mergeSortedArray(self, A, m, B, n):

i, j = m-1, n-1

t = len(A)-1

while i >= 0 or j >= 0:

if i < 0 or (j >= 0 and B[j] > A[i]):

A[t] = B[j]

j -= 1

else:

A[t] = A[i]

i -= 1

t -= 1

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

A = [1,2,3,0,0]

m = 3

B = [4,5]

n = 2

solution.mergeSortedArray(A, m, B, n)

print("输入：A = [1,2,3,0,0], 3, B = [4,5], 2")

print("输出：", A)

【例43】最小路径和

3.代码实现

class Solution:

#参数grid: 二维整数数组

#返回值: 一个整数，使其路径上的所有数字之和最小化

def minPathSum(self, grid):

for i in range(len(grid)):

for j in range(len(grid[0])):

if i == 0 and j > 0:

grid[i][j] += grid[i][j-1]

elif j == 0 and i > 0:

grid[i][j] += grid[i-1][j]

elif i > 0 and j > 0:

grid[i][j] += min(grid[i-1][j], grid[i][j-1])

return grid[len(grid) - 1][len(grid[0]) - 1]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

grid = [[1,3,1],[1,5,1],[4,2,1]]

length = solution.minPathSum(grid)

print("输入：", grid)

print("输出：", length)

【例44】大小写转换

3.代码实现

class Solution:

#参数character: 字符

#返回值: 字符

def lowercaseToUppercase(self, character):

#ASCII码中小写字母与对应的大写字母相差32

return chr(ord(character) - 32)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

ans = solution.lowercaseToUppercase('a')

print("输入： a")

print("输出：", ans)

【例45】原子的数量

3.代码实现

import re

import collections

class Solution(object):

def countOfAtoms(self, formula):

parse = re.findall(r"([A-Z][a-z]\*)(\d\*)|(\()|(\))(\d\*)", formula)

stack = [collections.Counter()]

for name, m1, left\_open, right\_open, m2 in parse:

if name:

stack[-1][name] += int(m1 or 1)

if left\_open:

stack.append(collections.Counter())

if right\_open:

top = stack.pop()

for k in top:

stack[-1][k] += top[k] \* int(m2 or 1)

return "".join(name + (str(stack[-1][name]) if stack[-1][name] > 1 else '') for name in sorted(stack[-1]))

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

Test\_in = "H2O"

Test\_out = solution.countOfAtoms(Test\_in)

print("输入为：",Test\_in)

print("输出为：",Test\_out)

【例46】矩阵中的最长递增路径

3.代码实现

DIRECTIONS = [(1, 0), (-1, 0), (0, -1), (0, 1)]

class Solution:

"""

参数matrix为整数矩阵

返回整数

"""

def longestIncreasingPath(self, matrix):

if not matrix or not matrix[0]:

return 0

sequence = []

for i in range(len(matrix)):

for j in range(len(matrix[0])):

sequence.append((matrix[i][j], i, j))

sequence.sort()

check = {}

for h, x, y in sequence:

cur\_pos = (x, y)

if cur\_pos not in check:

check[cur\_pos] = 1

cur\_path = 0

for dx, dy in DIRECTIONS:

if self.is\_valid(x+dx, y+dy, matrix, h):

cur\_path = max(cur\_path, check[(x+dx, y+dy)])

check[cur\_pos] += cur\_path

vals = check.values()

return max(vals)

def is\_valid(self, x, y, matrix, h):

row, col = len(matrix), len(matrix[0])

return x >= 0 and x < row and y >= 0 and y < col and matrix[x][y]<h

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

Test\_in = [

[9,9,4],

[6,6,8],

[2,1,1]

]

Test\_out = solution.longestIncreasingPath(Test\_in)

print("输入为：",Test\_in)

print("输出为：",Test\_out)

【例47】大小写转换

3.代码实现

class Solution:

#参数str: 字符串

#返回值: 字符串

def lowercaseToUppercase2(self, str):

p = list(str)

#遍历整个字符串，将所有的小写字母转成大写字母

for i in range(len(p)):

if p[i] >= 'a' and p[i] <= 'z':

p[i] = chr(ord(p[i]) - 32)

return ''.join(p)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

s1 = "abC12"

ans = solution.lowercaseToUppercase2(s1)

print("输入：", s1)

print("输出：", ans)

【例48】水仙花数

3.代码实现

class Solution:

#参数n: 数字的位数

@返回值: 所有n位数的水仙花数

def getNarcissisticNumbers(self, n):

res = []

for x in range([0, 10\*\*(n-1)][n > 1], 10\*\*n):

y, k = x, 0

while x > 0:

k += (x % 10)\*\*n

x //= 10

if k == y: res.append(k)

return res

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

n = 4

ans = solution.getNarcissisticNumbers(n)

print("输入：", n)

print("输出：", ans)

【例49】余弦相似度

3.代码实现

import math

#参数A，B都是一个整型数组，表示两个矢量

#返回值是两个输入矢量的余弦相似度

class Solution:

def cosineSimilarity(self, A, B):

if len(A) != len(B):

return 2

n = len(A)

up = 0

for i in range(n):

up += A[i] \* B[i]

down = sum(a\*a for a in A) \* sum(b\*b for b in B)

if down == 0:

return 2

return up / math.sqrt(down)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generator\_A = [1,4,0]

generator\_B = [1,2,3]

solution = Solution()

print("输入: A=", generator\_A)

print("输入: B=", generator\_B)

print("输出: ", solution.cosineSimilarity(generator\_A,generator\_B))

【例50】链表节点计数

3.代码实现

#参数head是链表的头部

#返回值是链表的长度

class ListNode(object):

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

self.val = val

self.next = next

class Solution:

def countNodes(self, head):

cnt = 0

while head is not None:

cnt += 1

head = head.next

return cnt

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

node1 = ListNode(1)

node2 = ListNode(2)

node3 = ListNode(3)

node4 = ListNode(4)

node1.next = node2

node2.next = node3

node3.next = node4

solution = Solution()

print("输入: ", node1.val,node2.val,node3.val,node4.val)

print("输出: ", solution. countNodes(node1))

【例51】最高频的K个单词

3.代码实现

#参数words是一个字符串数组

#参数k代表第k高频率出现

#返回值是一个字符串数组，表示出现频率前k高字符串

class Solution:

def topKFrequentWords(self, words, k):

dict = {}

res = []

for word in words:

if word not in dict:

dict[word] = 1

else:

dict[word] += 1

sorted\_d = sorted(dict.items(), key=lambda x:x[1], reverse=True)

for i in range(k):

res.append(sorted\_d[i][0])

return res

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generator = ["yes", "long", "code",

"yes", "code", "baby",

"you", "baby", "chrome",

"safari", "long", "code",

"body", "long", "code"]

k = 4

solution = Solution()

print("输入: ", generator)

print("输入: ","k = ", k)

print("输出: ", solution.topKFrequentWords(generator,k))

【例52】单词的添加与查找

3.代码实现

#参数word是要添加的的单词

#返回值是个布尔值，查找单词成功则返回True，否则，返回False

class TrieNode:

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

self.children = {}

self.is\_word = False

class WordDictionary:

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

self.root = TrieNode()

def addWord(self, word):

node = self.root

for c in word:

if c not in node.children:

node.children[c] = TrieNode()

node = node.children[c]

node.is\_word =True

def search(self, word):

if word is None:

return False

return self.search\_helper(self.root, word, 0)

def search\_helper(self, node, word, index):

if node is None:

return False

if index >= len(word):

return node.is\_word

char = word[index]

if char != '.':

return self.search\_helper(node.children.get(char), word, index + 1)

for child in node.children:

if self.search\_helper(node.children[child], word, index + 1):

return True

return False

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = WordDictionary()

solution.addWord("bad")

solution.addWord("dad")

solution.addWord("mad")

print('输入： addWord("bad"),addWord("dad"),addWord("mad")')

print('输入： search("pad"),search("dad"),search(".ad"),search("b..")')

print("输出： ",

solution.search("pad"),

solution.search("bad"),

solution.search(".ad"),

solution.search("b.."))

【例53】石子归并

3.代码实现

#参数A是一个整型数组

#返回值是一个整数，表示最小的合并代价

import sys

class Solution:

def stoneGame(self, A):

n = len(A)

if n < 2:

return 0

dp = [[0] \* n for \_ in range(n)]

cut = [[0] \* n for \_ in range(n)]

range\_sum = self.get\_range\_sum(A)

for i in range(n - 1):

dp[i][i + 1] = A[i] + A[i + 1]

cut[i][i + 1] = i

for length in range(3, n + 1):

for i in range(n - length + 1):

j = i + length - 1

dp[i][j] = sys.maxsize

for mid in range(cut[i][j - 1], cut[i + 1][j] + 1):

if dp[i][j] > dp[i][mid] + dp[mid + 1][j] + range\_sum[i][j]:

dp[i][j] = dp[i][mid] + dp[mid + 1][j] + range\_sum[i][j]

cut[i][j] = mid

return dp[0][n - 1]

def get\_range\_sum(self, A):

n = len(A)

range\_sum = [[0] \* n for \_ in range(len(A))]

for i in range(n):

range\_sum[i][i] = A[i]

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

range\_sum[i][j] = range\_sum[i][j - 1] + A[j]

return range\_sum

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generator = [3,4,3]

solution = Solution()

print("输入：", generator)

print("输出：", solution.stoneGame(generator))

【例54】简单计算器

3.代码实现

#参数a，b是两个任意整数

#operator是运算符+, -, \*, /

#返回值是浮点型运算结果

class Solution:

def calculate(self, a, operator, b):

if operator == '+':

return a + b

elif operator == '-':

return a - b

elif operator == '\*':

return a \* b

elif operator == '/':

return a / b

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

a=8

b=3

operator1='+'

operator2='-'

operator3='\*'

operator4='/'solution = Solution()

print("输入：", a ,operator1 ,b)

print("输出：", solution.calculate(a,operator1,b))

print("输入：", a ,operator2 ,b)

print("输出：", solution.calculate(a,operator2,b))

print("输入：", a ,operator3 ,b)

print("输出：", solution.calculate(a,operator3,b))

print("输入：", a ,operator4 ,b)

print("输出：", solution.calculate(a,operator4,b))

【例55】数组第二大数

3.代码实现

#参数nums是一个整型数组

#返回值secValue是数组中第二大数

class Solution:

def secondMax(self, nums):

maxValue = max(nums[0], nums[1])

secValue = min(nums[0], nums[1])

for i in range(2, len(nums)):

if nums[i] > maxValue:

secValue = maxValue

maxValue = nums[i]

elif nums[i] > secValue:

secValue = nums[i]

return secValue

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generator = [3,4,7,9]

solution = Solution()

print("输入： ", generator)

print("输出： ", solution.secondMax(generator))

【例56】二叉树叶子节点之和

3.代码实现

#参数root是二叉树的根

#返回值是个整数，叶子节点之和

class TreeNode:

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

self.val = val

self.left, self.right = None, None

class Solution:

def leafSum(self, root):

p = []

self.dfs(root, p)

return sum(p)

def dfs(self, root, p):

if root is None:

return

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

p.append(root.val)

self.dfs(root.left, p)

self.dfs(root.right, p)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

root = TreeNode(1)

root.left = TreeNode(2)

root.right = TreeNode(3)

root.left.left = TreeNode(4)

solution = Solution()

print("输入：", root.val,root.left.val,root.right.val,root.left.left.val)

print("输出：", solution.leafSum(root))

【例57】二叉树的某层节点之和

3.代码实现

#参数root是二叉树的根

#参数level是树的目标层的深度

#返回值是一个整数，表示该level叶子节点之和

class TreeNode:

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

self.val = val

self.left, self.right = None, None

class Solution:

def levelSum(self, root, level):

p = []

self.dfs(root, p, 1, level)

return sum(p)

def dfs(self, root, p, dep, level):

if root is None:

return

if dep == level:

p.append(root.val)

return

self.dfs(root.left, p, dep+1, level)

self.dfs(root.right, p, dep+1, level)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

root = TreeNode(1)

root.left = TreeNode(2)

root.right = TreeNode(3)

root.left.left = TreeNode(4)

root.left.right = TreeNode(5)

root.right.left = TreeNode(6)

root.right.right = TreeNode(7)

root.left.right.right = TreeNode(8)

root.right.right.right = TreeNode(9)

depth = 3

solution = Solution()

print("输入：",root.val,root.left.val,root.right.val,root.left.left.val,

root.left.right.val,root.right.left.val,root.right.right.val,

root.left.right.right.val,root.right.right.right.val)

print("输入： depth= ", depth)

print("输出：",solution.levelSum(root,depth))

【例58】判断尾数

3.代码实现

#参数str是输入01串

#返回值是一个整数，代表最后一个词的长度

class Solution:

def judgeTheLastNumber(self, str):

if str[-1] == 1:

return 2

for i in range(-2, -len(str) - 1, -1):

if str[i] == 0:

return -1 \* ((i \* -1 + 1) % 2) + 2

return -1 \* (len(str) % 2) + 2

if \_\_name\_\_ == '\_\_main\_\_':

str = "111110"

solution = Solution()

print(" 原01串为：", str)

print(" 最后一个词长度是:", solution.judgeTheLastNumber(str))

【例59】两个字符串是变位词

3.代码实现

class Solution:

#参数s: 第一个字符串

#参数t: 第二个字符串

#返回值: True或False

def anagram(self, s, t):

set\_s = [0] \* 256

set\_t = [0] \* 256

for i in range(0, len(s)):

set\_s[ord(s[i])] += 1

for i in range(0, len(t)):

set\_t[ord(t[i])] += 1

for i in range(0, 256):

if set\_s[i] != set\_t[i]:

return False

return True

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

s = "abcd"

t = "dcba"

ans = solution.anagram(s, t)

print("输入：", s, t)

print("输出：", ans)

【例60】最长单词

3.代码实现

class Solution:

#参数dictionary: 字符串数组

#返回值: 字符串数组

def longestWords(self, dictionary):

answer = []

maxLength = 0

for item in dictionary:

if len(item) > maxLength:

maxLength = len(item)

answer = [item]

elif len(item) == maxLength:

answer.append(item)

return answer

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

dic = ["dog","google","facebook","internationalization","blabla"]

answer = solution.longestWords(dic)

print("输入：", dic)

print("输出：", answer)

【例61】机器人能否返回原点

3.代码实现

class Solution:

#参数moves为字符串

#返回布尔类型

def judgeCircle(self, moves):

count\_RL = count\_UD = 0

for c in moves:

if c == 'R':

count\_RL += 1

if c == 'L':

count\_RL -= 1

if c == 'U':

count\_UD += 1

if c == 'D':

count\_UD -= 1

return count\_RL == 0 and count\_UD == 0

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

moves="UD"

print("输入为：",moves)

print("输出为：",solution.judgeCircle(moves))

【例62】链表倒数第n个节点

3.代码实现

#定义链表节点

class ListNode(object):

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

self.val = val

self.next = None

class Solution:

#参数head: 链表的第一个节点。

#参数n: 整数

#返回值: 单链表的第n到最后一个节点。

def nthToLast(self, head, n):

if head is None or n < 1:

return None

cur = head.next

while cur is not None:

cur.pre = head

cur = cur.next

head = head.next

n -= 1

while n > 0:

head = head.pre

n -= 1

return head

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

l0 = ListNode(3)

l1 = ListNode(2)

l2 = ListNode(1)

l3 = ListNode(5)

l0.next = l1

l1.next = l2

l2.next = l3

ans = solution.nthToLast(l0, 2).val

print("输入： 3->2->1->5->null, n = 2")

print("输出：", ans)

【例63】链表求和

3.代码实现

#定义链表节点

class ListNode(object):

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

self.val = val

self.next = None

class Solution:

def addLists(self, l1, l2) -> list:

dummy = ListNode(None)

tail = dummy

carry = 0

while l1 or l2 or carry:

num = 0

if l1:

num += l1.val

l1 = l1.next

if l2:

num += l2.val

l2 = l2.next

num += carry

digit, carry = num % 10, num // 10

node = ListNode(digit)

tail.next, tail = node, node

return dummy.next

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

l0 = ListNode(7)

l1 = ListNode(1)

l2 = ListNode(6)

l0.next = l1

l1.next = l2

l3 = ListNode(5)

l4 = ListNode(9)

l5 = ListNode(2)

l3.next = l4

l4.next = l5

ans = solution.addLists(l0, l3)

a = [ans.val, ans.next.val, ans.next.next.val]

print("输入： 7->1->6->null, 5->9->2->null")

print("输出： 2->1->9->null")

【例64】删除元素

3.代码实现

class Solution:

#参数A: 整数列表

#参数elem: 整数

#返回值：移除后的长度

def removeElement(self, A, elem):

j = len(A)-1

for i in range(len(A) - 1, -1, -1):

if A[i] == elem:

A[i], A[j] = A[j], A[i]

j -= 1

return j+1

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

A = [0,4,4,0,0,2,4,4]

e = 4

ans = solution.removeElement(A, e)

print("输入： [0,4,4,0,0,2,4,4], value = 4")

print("输出：", ans)

【例65】克隆二叉树

3.代码实现

#树的节点结构

#参数val是节点值

class TreeNode:

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

self.val = val

self.left, self.right = None, None

#参数{TreeNode} root是二进制树的根

#返回值clone\_root是复制后新树的根

class Solution:

def cloneTree(self, root):

if root is None:

return None

clone\_root = TreeNode(root.val)

clone\_root.left = self.cloneTree(root.left)

clone\_root.right = self.cloneTree(root.right)

return clone\_root

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

root = TreeNode(1)

root.left = TreeNode(2)

root.right = TreeNode(3)

root.left.left = TreeNode(4)

root.left.right = TreeNode(5)

solution = Solution()

print("输入：", root.val,root.left.val,root.right.val,root.left.left.val,root.left.right.val)

print("输出： ", solution.cloneTree(root).val,solution.cloneTree(root).left.val,solution.cloneTree(root).right.val,solution.cloneTree(root).left.left.val,solution.cloneTree(root).left.right.val)

【例66】合并两个排序链表

3.代码实现

#定义链表节点

class ListNode(object):

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

self.val = val

self.next = None

class Solution(object):

#参数l1: 链表头结节点

#参数l2: 链表头节点

#返回值: 链表头节点

def mergeTwoLists(self, l1, l2):

dummy = ListNode(0)

tmp = dummy

while l1 != None and l2 != None:

if l1.val < l2.val:

tmp.next = l1

l1 = l1.next

else:

tmp.next = l2

l2 = l2.next

tmp = tmp.next

if l1 != None:

tmp.next = l1

else:

tmp.next = l2

return dummy.next

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

l0 = ListNode(1)

l1 = ListNode(3)

l2 = ListNode(8)

l0.next = l1

l1.next = l2

l5 = ListNode(2)

l6 = ListNode(4)

l5.next = l6

ans = solution.mergeTwoLists(l0, l5)

a = [ans.val, ans.next.val, ans.next.next.val,

ans.next.next.next.val, ans.next.next.next.next.val]

print("输入： list1 = 1->3->8->null, list2 = 2->4->null")

print("输出： 1->2->3->4->8->null")

【例67】反转整数

3.代码实现

#参数n是一个整型数

#返回值reverse是反转的整数

class Solution:

def reverseInteger(self, n):

if n == 0:

return 0

neg = 1

if n < 0:

neg, n = -1, -n

reverse = 0

while n > 0:

reverse = reverse \* 10 + n % 10

n = n // 10

reverse = reverse \* neg

if reverse < -(1 << 31) or reverse > (1 << 31) - 1:

return 0

return reverse

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generator=1234

solution = Solution()

print("输入：", generator)

print("输出：", solution. reverseInteger(generator))

【例68】报数

3.代码实现

#参数n是一个正整数

#返回值string是n所表示的报数序列

class Solution:

def countAndSay(self, n):

string = '1'

for i in range(n - 1):

a = string[0]

count = 0

s = ''

for ch in string:

if a == ch:

count += 1

else:

s += str(count) + a

a = ch

count = 1

s += str(count) + a

string = s

a = string[0]

return string

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generator=5

solution = Solution()

print("输入:", generator)

print("输出:", solution.countAndSay(generator))

【例69】完全二叉树

3.代码实现

#参数root是二叉树的根

#返回值是个布尔值，当完全二叉树时返回True，否则返回False

class TreeNode:

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

self.val = val

self.left = None

self.right = None

class Solution:

def isComplete(self, root):

if root is None:

return True

queue = [root]

index = 0

while index < len(queue):

if queue[index] is not None:

queue.append(queue[index].left)

queue.append(queue[index].right)

index += 1

while queue[-1] is None:

queue.pop()

for q in queue:

if q is None:

return False

return True

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

root = TreeNode(1)

root.left = TreeNode(2)

root.right = TreeNode(3)

root.left.left = TreeNode(4)

solution = Solution()

print("输入: ", root.val,root.left.val,root.right.val,root.left.left.val)

print("输出: ", solution.isComplete(root))

【例70】对称二叉树

3.代码实现

#参数root是二叉树的的根

#返回值是个布尔值，是对称二叉树时返回True，否则返回False

class TreeNode:

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

self.val = val

self.left = None

self.right = None

class Solution:

def help(self, p, q):

if p == None and q == None: return True

if p and q and p.val == q.val:

return self.help(p.right, q.left) and self.help(p.left, q.right)

return False

def isSymmetric(self, root):

if root:

return self.help(root.left, root.right)

return True

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

root = TreeNode(1)

root.left = TreeNode(2)

root.right = TreeNode(2)

root.right.right = TreeNode(3)

root.right.left = TreeNode(4)

root.left.right = TreeNode(4)

root.left.left = TreeNode(3)

solution = Solution()

print("输入: ",

root.val,root.left.val,root.right.val,root.left.left.val,root.left.right. val,root.right.left.val, root.right.right.val)

print("输出: ", solution.isSymmetric(root))

【例71】扭转后等价的二叉树

3.代码实现

#参数a、b是二叉树的根

#返回值是个布尔值，当它们等价时返回True，否则返回False

class TreeNode:

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

self.val = val

self.left = None

self.right = None

class Solution:

def isTweakedIdentical(self, a, b):

if a == None and b == None: return True

if a and b and a.val == b.val:

return self.isTweakedIdentical(a.left, b.left) and \

self.isTweakedIdentical(a.right, b.right) or \

self.isTweakedIdentical(a.left, b.right) and \

self.isTweakedIdentical(a.right, b.left)

return False

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

root = TreeNode(1)

root.left = TreeNode(2)

root.right = TreeNode(3)

root.left.left = TreeNode(4)

root1 = TreeNode(1)

root1.right = TreeNode(2)

root1.left = TreeNode(3)

root1.right.right = TreeNode(4)

solution = Solution()

print("输入: ", root.val,root.left.val,root.right.val,root.left.left.val," , ",root1.val,root1.left.val,root1.right.val,root1.right.right.val)

print("输出: ", solution.isTweakedIdentical(root,root1))

【例72】岛屿的个数

3.代码实现

from collections import deque

#参数grid是一个01矩阵

#返回值islands是岛屿的个数

class Solution:

def numIslands(self, grid):

if not grid or not grid[0]:

return 0

islands = 0

for i in range(len(grid)):

for j in range(len(grid[0])):

if grid[i][j]:

self.bfs(grid, i, j)

islands += 1

return islands

def bfs(self, grid, x, y):

queue = deque([(x, y)])

grid[x][y] = False

while queue:

x, y = queue.popleft()

for delta\_x, delta\_y in [(1, 0), (0, -1), (-1, 0), (0, 1)]:

next\_x = x + delta\_x

next\_y = y + delta\_y

if not self.is\_valid(grid, next\_x, next\_y):

continue

queue.append((next\_x, next\_y))

grid[next\_x][next\_y] = False

def is\_valid(self, grid, x, y):

n, m = len(grid), len(grid[0])

return 0 <= x < n and 0 <= y < m and grid[x][y]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generator= [

[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]

]

solution = Solution()

print("输入：", generator)

print("输出：", solution.numIslands(generator))

【例73】判断是否为平方数之和

3.代码实现

import math

class Solution:

"""

参数num为整数

返回布尔类型

"""

def checkSumOfSquareNumbers(self, num):

# write your code here

if num < 0:

return False

for i in reversed(range(0, int(math.sqrt(num)) + 1)):

if i \* i == num:

return True

j = num - i \* i

k = int(math.sqrt(j))

if k \* k == j:

return True

return False

if \_\_name\_\_=='\_\_main\_\_':

solution=Solution()

num=5

print("输入为：",num)

print("输出为：",solution.checkSumOfSquareNumbers(num))

【例74】滑动窗口内数的和

3.代码实现

class Solution:

#nums是整数数组

#k是滑动窗口

#返回每个窗口的数字和

def winSum(self, nums, k):

n = len(nums)

if n < k or k <= 0:

return []

sums = [0] \* (n - k + 1)

for i in range(k):

sums[0] += nums[i];

for i in range(1, n - k + 1):

sums[i] = sums[i - 1] - nums[i - 1] + nums[i + k - 1]

return sums

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

inputnum=[1,2,7,8,5]

k=3

print("输入数组：",inputnum)

print("输入窗口：",k)

solution=Solution()

print("输出数组：",solution.winSum(inputnum,k))

4.运行结果

输入数组：[1, 2, 7, 8, 5]

输入窗口：3

输出数组：[10, 17, 20]

【例75】棒球游戏

【例76】硬币摆放

3.代码实现

import math

class Solution:

#参数n为整数

#返回整数

# n = (1 + x) \* x / 2, 求得 x = (-1 + sqrt(8 \* n + 1)) / 2, 对x取整

def arrangeCoins(self, n):

return math.floor((-1 + math.sqrt(1 + 8\*n)) / 2)

if \_\_name\_\_ == '\_\_main\_\_':

n = 10

solution = Solution()

print("输入为：",n)

print("输出为：",solution.arrangeCoins(n))

【例77】字母大小写转换

3.代码实现

class Solution(object):

def letterCasePermutation(self, S):

#参数S为字符串

#返回字符串列表

# 利用二进制对应字符串。其中0表示大小写不变，1表示改变大小写

res = []

indices = []

indices = [i for i,\_ in enumerate(S) if S[i].isalpha()]

for i in range(0, pow(2,len(indices))):

if i==0:

res.append(S)

else:

j=i;bpos=0;nsl=list(S)

while j>0:

ci2c = indices[bpos]

if j&1 and S[ci2c].islower():

nsl[ci2c]=S[ci2c].upper()

elif j&1 and S[ci2c].isupper():

nsl[ci2c]=S[ci2c].lower()

bpos+=1

j = j >> 1

res.append("".join(nsl))

return res

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

S = "a1b2"

print("输入为：",S)

print("输出为：",solution.letterCasePermutation(S))

【例78】二进制表示中质数个计算置位

3.代码实现

class Solution(object):

def countPrimeSetBits(self, L, R):

# "L, R在[1, 10^6]范围

# 可能的质数为2, 3, 5, 7, 11, 13, 17, 19.

# 统计1的个数在进行质数判定,因为二进制1的个数不会超过20个，枚举质数即可

k = 0

for n in range(L, R + 1):

if bin(n).count('1') in [2, 3, 5, 7, 11, 13, 17, 19]:

k = k + 1

return k

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

L=6

R=10

print("输入为：[",L,R,"]")

print("输出为：",solution.countPrimeSetBits(L,R))

【例79】最少费用的爬台阶方法

3.代码实现

class Solution:

#参数cost为数组

#返回最小费用

#状态转移方程 dp[i] = min(dp[i-1] + cost[i-1],dp[i-2] + cost[i-2])

def minCostClimbingStairs(self, cost):

a, b = 0, 0

for i in range(2, len(cost) + 1):

c = min(a + cost[i - 2], b + cost[i - 1])

a, b = b, c

return b

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

print("输入为：",cost)

print("输出为：",solution.minCostClimbingStairs(cost))

【例80】中心索引

3.代码实现

class Solution(object):

def pivotIndex(self, nums):

left, right = 0, sum(nums)

for index, num in enumerate(nums):

right -= num

if left == right:

return index

left += num

return -1

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

words=[1,7,3,6,5,6]

print(solution.pivotIndex(words))

【例81】词典中最长的单词

3.代码实现

class Solution(object):

def longestWord(self, words):

words.sort()

words.sort(key=len, reverse=True)

res = []

for word in words:

temp = word

i = 1

for i in range(len(temp)):

if temp[:len(temp) - i] in words:

if i == len(temp) - 1:

return temp

continue

else:

break

return ''

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

words=["w","wo","wor","worl", "world"]

print("输入字典为：",words)

print("输出单词为：",solution.longestWord(words))

【例82】重复字符串匹配

3.代码实现

class Solution:

#参数A为字符串

#参数B为字符串

#返回整数

def repeatedStringMatch(self, A, B):

C = ""

for i in range(int(len(B)/len(A) + 3)):

if B in C:

return i

C += A

return -1

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

A = "abcd"

B = "cdabcdab"

print("输入字符串A为：",A)

print("输入字符串B为：",B)

print("需要重复次数：",solution.repeatedStringMatch(A,B))

【例83】不下降数组

3.代码实现

class Solution:

#参数nums为数组

#返回布尔类型

def checkPossibility(self, nums):

count = 0

for i in range(1, len(nums)):

if nums[i] < nums[i - 1]:

count += 1

if i >= 2 and nums[i] < nums[i - 2]:

nums[i] = nums[i - 1]

else:

nums[i - 1] = nums[i]

return count <= 1

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

nums=[4,2,3]

print("输入为：",nums)

print("输出为：",solution.checkPossibility(nums))

【例84】最大的回文乘积

3.代码实现

class Solution:

#参数n为整数

#返回整数

def largestPalindrome(self, n):

if n==1:

return 9

elif n ==7:

return 877

elif n== 8:

return 475

maxNum,minNum = 10\*\*n - 1, 10\*\*(n-1)

for i in range(maxNum, minNum, -1):

candidate = str(i)

candidate = candidate + candidate [::-1]

candidate = int(candidate)

j = maxNum

while j\*j > candidate :

if candidate % j == 0:

return candidate % 1337

j -= 1

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n = 2

print("输入为：",n)

print("输出为：",s.largestPalindrome(n))

【例85】补数

3.代码实现

class Solution:

#参数num为整数

#返回整数

def findComplement(self, num):

return num ^ ((1<<num.bit\_length())-1)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n = 5

print("输入为：",n)

print("输出为：",s.findComplement(n))

【例86】加热器

3.代码实现

class Solution:

#参数houses为数组

#参数heaters为整数

#返回整数

def findRadius(self, houses, heaters):

heaters.sort()

ans = 0

for house in houses:

ans=max(ans,self.closestHeater(house,heaters))

return ans

def closestHeater(self,house,heaters):

start = 0

end = len(heaters) - 1

while start + 1 < end:

m = start + (end - start) // 2

if heaters[m] == house:

return 0

elif heaters[m] < house:

start = m

else:

end = m

return min(abs(house - heaters[start]), abs(heaters[end] - house))

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n = [1,2,3]

m = [2]

print("输入房间位置为：",n)

print("输入加热器位置：",m)

print("输出加热半径为：",s.findRadius(n,m))

【例87】将火柴摆放成正方形

3.代码实现

class Solution:

#参数nums为数组

#返回布尔类型

def makesquare(self, nums):

def dfs(nums, pos, target):

if pos == len(nums):

return True

for i in range(4):

if target[i] >= nums[pos]:

target[i] -= nums[pos]

if dfs(nums, pos+1, target):

return True

target[i] += nums[pos]

return False

if len(nums) < 4 :

return False

numSum = sum(nums)

nums.sort(reverse = True)

if numSum % 4 != 0:

return False

target = [numSum / 4] \* 4;

return dfs(nums, 0, target)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n = [1,1,2,2,2]

print("输入为：",n)

print("输出为：",s.makesquare(n))

【例88】可怜的猪

3.代码实现

class Solution:

#参数buckets为整数

#参数minutesToDie为整数

#参数minutesToTest为整数

返回整数

def poorPigs(self, buckets, minutesToDie, minutesToTest):

pigs = 0

while (minutesToTest / minutesToDie + 1) \*\* pigs < buckets:

pigs += 1

return pigs

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n = 1000

m = 15

p = 60

print("输入总桶数为：",n)

print("输入中毒时间：",m)

print("输入测试时间：",p)

print("输出为：",s.poorPigs(n,m,p))

【例89】循环数组中的环

3.代码实现

class Solution:

#参数nums为数组

#返回布尔类型

def get\_index(self, i, nums):

n = (i + nums[i]) % len(nums)

return n if n >= 0 else n + len(nums)

def circularArrayLoop(self, nums):

for i in range(len(nums)):

if nums[i] == 0:

continue

j, k = i, self.get\_index(i, nums)

while nums[k] \* nums[i] > 0 and nums[self.get\_index(k, nums)] \* nums[i] > 0:

if j == k:

if j == self.get\_index(j, nums):

break

return True

j = self.get\_index(j, nums)

k = self.get\_index(self.get\_index(k, nums), nums)

j = i

while nums[j] \* nums[i] > 0:

next = self.get\_index(j, nums)

nums[j] = 0

j = next

return False

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n = [2,-1,1,2,2]

print("输入为：",n)

print("输出为：",s.circularArrayLoop(n))

【例90】分饼干

3.代码实现

class Solution(object):

def findContentChildren(self, g, s):

#参数g为整数列表

#参数s为整数列表

#返回整型

g.sort()

s.sort()

i, j = 0, 0

while i < len(g) and j < len(s):

if g[i] <= s[j]:

i += 1

j += 1

return i

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n = [1,2,3]

m = [1,1]

print("输入贪吃指数为：",n)

print("输入饼干尺寸为：",m)

print("输出为：",s.findContentChildren(n,m))

【例91】翻转字符串中的元音字母

3.代码实现

class Solution:

"""

参数s为字符串

返回字符串

"""

def reverseVowels(self, s):

vowels = set(["a", "e", "i", "o", "u", "A", "E", "I", "O", "U"])

res = list(s)

start, end = 0, len(res)-1

while start <= end:

while start <= end and res[start] not in vowels:

start += 1

while start <= end and res[end] not in vowels:

end -= 1

if start <= end:

res[start], res[end] = res[end], res[start]

start += 1

end -= 1

return "".join(res)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

x = "hello"

print("输入为：",x)

print("输出为：",s.reverseVowels(x))

【例92】翻转字符串

3.代码实现

class Solution:

"""

参数s为字符串

返回字符串

"""

def reverseString(self, s):

return s[::-1]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

x = "hello"

print("输入为：",x)

print("输出为：",s.reverseString(x))

【例93】使数组元素相同的最少步数

3.代码实现

class Solution(object):

def minMoves(self, nums):

#参数nums为整数列表

#返回整数

sumNum = sum(nums)

minNum = min(nums)

return sumNum - minNum \* len(nums)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n = [1,2,3]

print("输入为：",n)

print("输出为：",s.minMoves(n))

【例94】加油站

3.代码实现

#参数distance代表每个加油站的距离

#参数apply代表每个加油站的加油量

#参数original代表一开始有汽油

#参数target代表需要开的距离

#返回值是一个整数，代表至少需要加油站次数

class Solution:

def getTimes(self, target, original, distance, apply):

import queue

que = queue.PriorityQueue()

ans, pre = 0, 0

if(target > distance[len(distance) - 1]):

distance.append(target)

apply.append(0)

cap = original

for i in range(len(distance)):

if(distance[i] >= target):

distance[i] = target

d = distance[i] - pre

while(cap < d and que.qsize() != 0):

cap += (que.get()[1])

ans += 1

if (d <= cap):

cap -= d

else:

ans = -1

break

que.put((-apply[i], apply[i]))

pre = distance[i]

if(pre == target):

break

return ans

if \_\_name\_\_ == '\_\_main\_\_':

target = 25

original = 10

distance = [10,14,20,21]

apply = [10,5,2,4]

solution = Solution()

print(" 每个加油站的距离为：", distance)

print(" 每个加油站的加油量为：", apply)

print(" 一开始有汽油：", original)

print(" 需要开的距离为：", target)

print(" 至少需要经过加油站:", solution.getTimes(target, original, distance, apply))

【例95】春游

3.代码实现

#参数a是小朋友组链

#返回值是个整数，表示至少需要多少辆车

class Solution:

def getAnswer(self, a):

count = [0 for i in range(0, 5)]

for i in range(0, len(a)):

count[a[i]] = count[a[i]] + 1

count[1] = count[1] - count[3]

if count[2] % 2 == 1:

count[2] = count[2] + 1

count[1] = count[1] - 2

res = count[4] + count[3] + count[2] / 2

if count[1] > 0:

res = res + count[1] / 4

if not count[1] % 4 == 0:

res = res + 1

return int(res)

if \_\_name\_\_ == '\_\_main\_\_':

a = [1,2,3,4]

solution = Solution()

print(" 小朋友分组为：", a)

print(" 至少需要：", solution.getAnswer(a), "辆车")

【例96】合法数组

3.代码实现

#参数a是待查数组

#返回值是一个数值，代表出现奇数次的值或者数组不合法

class Solution:

def isValid(self, a):

countSet = {}

for i in a:

if i in countSet:

countSet[i] = countSet[i] + 1

else:

countSet[i] = 1

isHas = False

for key in countSet:

if countSet[key] % 2 == 1:

if isHas:

return -1

else:

isHas = True

ans = key

if isHas:

return ans

return -1

if \_\_name\_\_ == '\_\_main\_\_':

a=[1,1,2,2,3,3,4,4,5,5]

solution = Solution()

print(" 数组为：", a)

ans = solution.isValid(a)

print(" 数组奇数个的值是:" if ans != -1 else " 数组不合法", ans)

【例97】删除排序数组中的重复数字

3.代码实现

class Solution:

#参数A: 整数列表

#返回值：整数

def removeDuplicates(self, A):

B = []

before = None

countb = 0

for number in A:

if(before != number):

B.append(number)

before = number

countb = 1

elif countb < 2:

B.append(number)

countb += 1

p = 0

for number in B:

A[p] = number

p += 1

return p

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

A = [1,1,1,2,2,3]

p = solution.removeDuplicates(A)

print("输入：", A)

print("输出：", p)

【例98】字符串的不同排列

3.代码实现

class Solution:

#参数str: 一个字符串

#返回值: 所有排列

def stringPermutation2(self, str):

result = []

if str == '':

return ['']

s = list(str)

s.sort()

while True:

result.append(''.join(s))

s = self.nextPermutation(s)

if s is None:

break

return result

def nextPermutation(self, num):

n = len(num)

i = n - 1

while i >= 1 and num[i - 1] >= num[i]:

i -= 1

if i == 0: return None

j = n - 1

while j >= 0 and num[j] <= num[i - 1]:

j -= 1

num[i - 1], num[j] = num[j], num[i - 1]

num[i:] = num[i:][::-1]

return num

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

s1 = "aabb"

ans = solution.stringPermutation2(s1)

print("输入：", s1)

print("输出：", ans)

【例99】全排列

3.代码实现

class Solution:

#参数nums: 一个整数列表

#返回值: 排列后的列表

def permute(self, nums):

def \_permute(result, temp, nums):

if nums == []:

result += [temp]

else:

for i in range(len(nums)):

\_permute(result, temp + [nums[i]], nums[:i] + nums[i+1:])

if nums is None:

return []

if nums is []:

return [[]]

result = []

\_permute(result, [], sorted(nums))

return result

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

nums = [1,2,3]

solution = Solution()

result = solution.permute(nums)

print("输入：", nums)

print("输出：", result)

【例100】带重复元素的排列

3.代码实现

class Solution:

#参数nums: 整数数组

#返回值: 唯一排列的列表。

def permuteUnique(self, nums):

def \_permute(result, temp, nums):

if nums == []:

result += [temp]

else:

for i in range(len(nums)):

if i > 0 and nums[i] == nums[i-1]:

continue

\_permute(result, temp + [nums[i]], nums[:i] + nums[i+1:])

if nums is None:

return []

if len(nums) == 0:

return [[]]

result = []

\_permute(result, [], sorted(nums))

return result

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

nums = [1,2,2]

result = solution.permuteUnique(nums)

print("输入：", nums)

print("输出：", result)

【例101】插入区间

3.代码实现

class Interval(object):

def \_\_init\_\_(self, start, end):

self.start = start

self.end = end

def get(self):

str1 = "(" + str(self.start) + "," + str(self.end) + ")"

return str1

def equals(self, Intervalx):

if self.start == Intervalx.start and self.end == Intervalx.end:

return 1

else:

return 0

class Solution:

#参数intevals: 已排序的非重叠区间列表

#参数newInterval: 新的区间

#返回值: 一个新的排序非重叠区间列表与新的区间

def insert(self, intervals, newInterval):

results = []

insertPos = 0

for interval in intervals:

if interval.end < newInterval.start:

results.append(interval)

insertPos += 1

elif interval.start > newInterval.end:

results.append(interval)

else:

newInterval.start = min(interval.start, newInterval.start)

newInterval.end = max(interval.end, newInterval.end)

results.insert(insertPos, newInterval)

return results

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

interval1 = Interval(1,2)

interval2 = Interval(5,9)

interval3 = Interval(2,5)

results = solution.insert([interval1,interval2], interval3)

print("输入：[",interval1.get(),",", interval2.get(),"]","", interval3.get())

print("输出：[", results[0].get(), "]")

【例102】N皇后问题

3.代码实现

class Solution:

#参数n: 皇后的数量

#返回值: 不同解的总数

total = 0

n = 0

def attack(self, row, col):

for c, r in self.cols.items():

if c - r == col - row or c + r == col + row:

return True

return False

def search(self, row):

if row == self.n:

self.total += 1

return

for col in range(self.n):

if col in self.cols:

continue

if self.attack(row, col):

continue

self.cols[col] = row

self.search(row + 1)

del self.cols[col]

def totalNQueens(self, n):

self.n = n

self.cols = {}

self.search(0)

return self.total

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

solution.totalNQueens(4)

print("输入：", solution.n)

print("输出：", solution.total)

【例103】主元素

3.代码实现

class Solution:

#参数nums: 整数数组

#返回值: 主元素

def majorityNumber(self, nums):

nums.sort()

i=0;j=0

while i<=len(nums):

j=nums.count(nums[i])

if j>len(nums)//3:

return nums[i]

i+=j

return

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

nums = [99,2,99,2,99,3,3]

n = solution.majorityNumber(nums)

print("输入：", "[99,2,99,2,99,3,3]")

print("输出：", n)

【例104】字符大小写排序

3.代码实现

class Solution:

#参数chars: 需要排序的字母数组

def sortLetters(self, chars):

chars.sort(key=lambda c: c.isupper())

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

str1 = "abAcD"

arr = list(str1)

solution.sortLetters(arr)

print("输入：", str1)

print("输出：", ''.join(arr))

【例105】上一个排列

3.代码实现

class Solution:

#参数num: 整数列表

#参数: 整数列表

def previousPermuation(self, num):

for i in range(len(num)-2, -1, -1):

if num[i] > num[i+1]:

break

else:

num.reverse()

return num

for j in range(len(num)-1, i, -1):

if num[j] < num[i]:

num[i], num[j] = num[j], num[i]

break

for j in range(0, (len(num) - i)//2):

num[i+j+1], num[len(num)-j-1] = num[len(num)-j-1], num[i+j+1]

return num

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

num = [1,3,2,3]

num1 = solution.previousPermuation(num)

print("输入：", "[1,3,2,3]")

print("输出：", num1)

【例106】下一个排列

3.代码实现

class Solution:

#参数num: 整数列表

#返回值: 整数列表

def nextPermutation(self, num):

for i in range(len(num)-2, -1, -1):

if num[i] < num[i+1]:

break

else:

num.reverse()

return num

for j in range(len(num)-1, i, -1):

if num[j] > num[i]:

num[i], num[j] = num[j], num[i]

break

for j in range(0, (len(num) - i)//2):

num[i+j+1], num[len(num)-j-1] = num[len(num)-j-1], num[i+j+1]

return num

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

num = [1,3,2,3]

num1 = solution.nextPermutation(num)

print("输入：", "[1, 3, 2, 3]")

print("输出：", num1)

【例107】二叉树的层次遍历

3.代码实现

class TreeNode:

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

self.val=val

self.left=left #左子树

self.right=right #右子树

class Solution:

#参数root: 二叉树的根

#返回值:从底向上的层次序遍历

def levelOrderBottom(self, root):

self.results = []

if not root:

return self.results

q = [root]

while q:

new\_q = []

self.results.append([n.val for n in q])

for node in q:

if node.left:

new\_q.append(node.left)

if node.right:

new\_q.append(node.right)

q = new\_q

return list(reversed(self.results))

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

root=TreeNode(1,TreeNode(2),TreeNode(3))

results = solution.levelOrderBottom(root)

print("输入： {1,2,3}")

print("输出：", results)

【例108】最长公共子串

3.代码实现

class Solution:

#参数A, B: 两个字符串

#返回值: 最长公共子串的长度

def longestCommonSubstring(self, A, B):

ans = 0

for i in range(len(A)):

for j in range(len(B)):

l = 0

while i + l < len(A) and j + l < len(B) \

and A[i + l] == B[j + l]:

l += 1

if l > ans:

ans = l

return ans

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

A = "ABCD"

B = "CBCE"

ans = solution.longestCommonSubstring(A, B)

print("输入：","A =",A,"B =",B)

print("输出：", ans)

【例109】最近公共祖先

3.代码实现

#Definition of TreeNode:

class TreeNode:

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

self.val=val

self.left=left #左子树

self.right=right #右子树

class Solution:

#参数root: 二叉搜索树的根

#参数A: 二叉树的一个节点

#参数B: 二叉树的一个节点

#返回值: 返回两个节点的最低公共祖先(LCA)

def lowestCommonAncestor(self, root, A, B):

# A且其下面有B => A

# B且其下面有A => B

# A且其下面啥都没有 => A

# B且其下面啥都有 => B

if root is None:

return None

if root == A or root == B:

return root

left\_result = self.lowestCommonAncestor(root.left, A, B)

right\_result = self.lowestCommonAncestor(root.right, A, B)

# A 和 B 一边一个

if left\_result and right\_result:

return root

# 左子树有一个点或者左子树有LCA

if left\_result:

return left\_result

# 右子树有一个点或者右子树有LCA

if right\_result:

return right\_result

# 左右子树啥都没有

return None

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

tree = TreeNode(4, TreeNode(3), TreeNode(7, TreeNode(5), TreeNode(6)))

solution = Solution()

result = solution.lowestCommonAncestor(tree, tree.left, tree.right.left)

print("输入：{4,3,7,#,#,5,6}, LCA(3,5)")

print("输出：", result.val)

【例110】k数和

3.代码实现

class Solution:

def kSumII(self, A, k, target):

anslist = []

self.dfs(A, k, target, 0, [], anslist)

return anslist

def dfs(self, A, k, target, index, onelist, anslist):

if target == 0 and k == 0:

anslist.append(onelist)

return

if len(A) == index or target < 0 or k < 0:

return

self.dfs(A, k, target, index + 1, onelist, anslist)

self.dfs(A, k - 1, target - A[index], index + 1 , onelist + [A[index]], anslist)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

A = [1,2,3,4]

k = 2

target = 5

anslist = solution.kSumII(A, k, target)

print("输入：A = [1,2,3,4] k = 2 target = 5")

print("输出：", anslist)

【例111】有序链表转换为二分查找树

3.代码实现

#定义链表节点

class ListNode(object):

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

self.val = val

self.next = next

#定义树节点

class TreeNode:

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

self.val = val

self.left, self.right = None, None

class Solution:

#参数head: 链表的第一个节点

#返回值: 树节点

def sortedListToBST(self, head):

num\_list = []

while head:

num\_list.append(head.val)

head = head.next

return self.create(num\_list, 0, len(num\_list) - 1)

def create(self, nums, start, end):

if start > end:

return None

if start == end:

return TreeNode(nums[start])

root = TreeNode(nums[(start + end) // 2])

root.left = self.create(nums, start, (start + end) // 2 - 1) #注意是-1

root.right = self.create(nums, (start + end) // 2 + 1, end)

return root

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

listnode = ListNode(1, ListNode(2, ListNode(3)))

root = solution.sortedListToBST(listnode)

print("输入： 1=>2=>3")

print("输出：", "{", root.val, root.left.val, root.right.val, "}")

【例112】最长连续序列

3.代码实现

class Solution:

#参数num：整数数组

#返回值：整数

def longestConsecutive(self, num):

dict={}

for x in num:

dict[x] = 1

ans = 0

for x in num:

if x in dict:

len = 1

del dict[x]

l = x - 1

r = x + 1

while l in dict:

del dict[l]

l -= 1

len += 1

while r in dict:

del dict[r]

r += 1

len += 1

if ans < len:

ans = len

return ans

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

num = [100, 4, 200, 1, 3, 2]

ans = solution.longestConsecutive(num)

print("输入：", num)

print("输出：", ans)

【例113】背包问题

3.代码实现

class Solution:

#参数m: 整数m表示背包的大小

#参数A & V: 给定n个大小为A[i]和值V[i]的物品

def backPackII(self, m, A, V):

f = [0 for i in range(m+1)]

n = len(A)

for i in range(n):

for j in range(m, A[i]-1, -1):

f[j] = max(f[j] , f[j-A[i]] + V[i])

return f[m]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

m = 100

A = [77,22,29,50,99]

V = [92,22,87,46,90]

result = solution.backPackII(m, A, V)

print("输入：\n","m = ",m, "\n A = ", A, "\n V = ",V)

print("输出：", result)

【例114】拓扑排序

3.代码实现

#定义有向图节点

class DirectedGraphNode:

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

self.label = x

self.neighbors = []

class Solution:

#参数graph: 有向图节点列表

#返回值: 整数列表

def topSort(self, graph):

indegree = {}

for x in graph:

indegree[x] = 0

for i in graph:

for j in i.neighbors:

indegree[j] += 1

ans = []

for i in graph:

if indegree[i] == 0:

self.dfs(i, indegree, ans)

return ans

def dfs(self, i, indegree, ans):

ans.append(i.label)

indegree[i] -= 1

for j in i.neighbors:

indegree[j] -= 1

if indegree[j] == 0:

self.dfs(j, indegree, ans)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

g0 = DirectedGraphNode(0)

g1 = DirectedGraphNode(1)

g2 = DirectedGraphNode(2)

g3 = DirectedGraphNode(3)

g4 = DirectedGraphNode(4)

g5 = DirectedGraphNode(5)

g0.neighbors = [g1, g2, g3]

g1.neighbors = [g4]

g2.neighbors = [g4, g5]

g3.neighbors = [g4, g5]

graph = [g0, g1, g2, g3, g4, g5]

result = solution.topSort(graph)

print("输入：如样例图")

print("输出：",result)

【例115】克隆图

3.代码实现

#定义无向图节点

class UndirectedGraphNode:

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

self.label = x

self.neighbors = []

class Solution:

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

self.dict = {}

#参数node: 无向图节点

#返回值: 无向图节点

def cloneGraph(self, node):

if node is None:

return None

if node.label in self.dict:

return self.dict[node.label]

root = UndirectedGraphNode(node.label)

self.dict[node.label] = root

for item in node.neighbors:

root.neighbors.append(self.cloneGraph(item))

return root

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

g0 = UndirectedGraphNode(0)

g1 = UndirectedGraphNode(1)

g2 = UndirectedGraphNode(2)

g0.neighbors = [g1, g2]

g1.neighbors = [g2]

g2.neighbors = [g2]

ans = solution.cloneGraph(g0)

a = [ans.label, ans.neighbors[0].label, ans.neighbors[1].label,

ans.neighbors[0].neighbors[0].label, ans.neighbors[1].neighbors[0].label]

print("输入： {0,1,2#1,2#2,2}")

print("输出：", a)

【例116】不同的二叉查找树

3.代码实现

class Solution:

#参数n: 整数

#返回值: 整数

def numTrees(self, n):

dp = [1, 1, 2]

if n <= 2:

return dp[n]

else:

dp += [0 for i in range(n-2)]

for i in range(3, n + 1):

for j in range(1, i+1):

dp[i] += dp[j-1] \* dp[i-j]

return dp[n]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

n = 3

ans = solution.numTrees(n)

print("输入：", n)

print("输出：", ans)

【例117】汉诺塔

3.代码实现

class Solution:

def move(self, n, a, b, c, ans): #n为圆盘数，a代表初始位圆柱，b代表过渡位圆柱，c代表目标位圆柱

if n==1:

ans.append("from "+a+" to "+c)

else:

self.move(n-1,a,c,b,ans)

ans.append("from "+a+" to "+c)

self.move(n-1,b,a,c,ans)

return ans

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

ans = []

res = solution.move(3, 'A', 'B', 'C', ans)

print("输入： 3, 'A', 'B', 'C'")

print("输出：", res)

【例118】图中两个点之间的路线

3.代码实现

#定义有向图节点

class DirectedGraphNode:

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

self.label = x

self.neighbors = []

class Solution:

def dfs(self, i, countrd, graph, t):

if countrd[i] == 1:

return False

if i == t:

return True

countrd[i] = 1

for j in i.neighbors:

if countrd[j] == 0 and self.dfs(j, countrd, graph, t):

return True

return False

#参数graph: 有向图节点列表

#参数s: 起始有向图节点

#参数t: 终端有向图节点

#返回值: 布尔值

def hasRoute(self, graph, s, t):

countrd = {}

for x in graph:

countrd[x] = 0

return self.dfs(s, countrd, graph, t)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

gA = DirectedGraphNode('A')

gB = DirectedGraphNode('B')

gC = DirectedGraphNode('C')

gD = DirectedGraphNode('D')

gE = DirectedGraphNode('E')

gA.neighbors = [gB, gD]

gB.neighbors = [gC, gD]

gD.neighbors = [gE]

graph = [gA, gB, gC, gD, gE]

ans = solution.hasRoute(graph, gB, gE)

print("输入： {A,B,C,D,E,A#B,A#D,B#C,B#D,D#E}, B, E")

print("输出：", ans)

【例119】丢失的第一个正整数

3.代码实现

class Solution:

#参数A：整数数组

#返回值：整数

def firstMissingPositive(self, A):

n = len(A)

i = 0

if n == 0:

return 1

while i < n:

while A[i]!=i+ 1 and A[i] <= n and A[i] > 0 and A[i] != A[A[i] - 1]:

t = A[i]

A[i] = A[A[i] - 1]

A[t - 1] = t

i = i + 1

for i in range(n):

if A[i] != i + 1: return i + 1

return n + 1

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

A = [3,4,-1,1]

ans = solution.firstMissingPositive(A)

print("输入：", A)

print("输出：", ans)

【例120】寻找缺失的数

3.代码实现

class Solution:

def findMissing(self, nums):

if not nums:

return 0

sum = 0

for \_ in nums:

sum += \_

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

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

nums = [0,1,3]

ans = solution.findMissing(nums)

print("输入：", nums)

print("输出：", ans)

【例121】排列序号I

3.代码实现

class Solution:

#参数A: 整数数组

#返回值: 整数

def permutationIndex(self, A):

result = 1

factor = 1

for i in range(len(A)-1, -1, -1):

rank = 0

for j in range(i+1, len(A)):

if A[i] > A[j]:

rank +=1

result += factor\*rank

factor \*= len(A)-i

return result

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

A = [3,2,1]

ans = solution.permutationIndex(A)

print("输入：", A)

print("输出：", ans)

【例122】排列序号II

3.代码实现

class Solution:

#参数A: 整数数组

#返回值: 长整数

def permutationIndexII(self, A):

if A is None or len(A) == 0:

return 0

index, factor, multi\_fact = 1, 1, 1

counter = {}

for i in range(len(A) - 1, -1, -1):

counter[A[i]] = counter.get(A[i], 0) + 1

multi\_fact \*= counter[A[i]]

count = 0

for j in range(i + 1, len(A)):

if A[i] > A[j]:

count += 1

index += count \* factor // multi\_fact

factor \*= (len(A) - i)

return index

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

A = [1,4,2,2]

ans = solution.permutationIndexII(A)

print("输入：", A)

print("输出：", ans)

【例123】最多有k个不同字符的最长子字符串

3.代码实现

#参数s是一个字符串

#返回值res是最长字符串的长度

class Solution:

def lengthOfLongestSubstring(self, s):

res = 0

if s is None or len(s) == 0:

return res

d = {}

tmp = 0

start = 0

for i in range(len(s)):

if s[i] in d and d[s[i]] >= start:

start = d[s[i]] + 1

tmp = i - start + 1

d[s[i]] = i

res = max(res, tmp)

return res

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generator = 'eceba'

solution = Solution()

print("输入：", generator)

print("输出：", solution. lengthOfLongestSubstring(generator))

【例124】第k个排列

3.代码实现

#参数n是指1~n

#参数k是指所有全排列中的第几个

#返回值是第k个全排列

class Solution:

def getPermutation(self, n, k):

fac = [1]

for i in range(1, n + 1):

fac.append(fac[-1] \* i)

elegible = list(range(1, n + 1))

per = []

for i in range(n):

digit = (k - 1) // fac[n - i - 1]

per.append(elegible[digit])

elegible.remove(elegible[digit])

k = (k - 1) % fac[n - i - 1] + 1

return "".join([str(x) for x in per])

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

k=4

n=3

solution = Solution()

print("输入：", "n=",n,"k=",k)

print("输出：", solution.getPermutation(n,k))

【例125】数飞机

3.代码实现

#参数start是开始时间

#参数end是结束时间

class Interval(object):

def \_\_init\_\_(self, start, end):

self.start = start

self.end = end

#参数airplanes是一个由时间间隔对象组成的数组

#返回值max\_number\_of\_airplane是飞机的最大数

class Solution:

def countOfAirplanes(self, airplanes):

points = []

for airplane in airplanes:

points.append([airplane.start, 1])

points.append([airplane.end, -1])

number\_of\_airplane, max\_number\_of\_airplane = 0, 0

for \_, count\_delta in sorted(points):

number\_of\_airplane += count\_delta

max\_number\_of\_airplane = max(max\_number\_of\_airplane,

number\_of\_airplane)

return max\_number\_of\_airplane

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generator=[Interval(1,10),Interval(5,8),Interval(2,3),Interval(4,7)]

solution = Solution()

print("输入：",[(1, 10), (2, 3), (5, 8), (4, 7)] )

print("输出：", solution.countOfAirplanes(generator))

【例126】格雷编码

3.代码实现

#参数n是一个整型数

#返回值seq是所对应的格雷编码

class Solution：

def grayCode(self, n):

if n == 0:

return [0]

result = self.grayCode(n - 1)

seq = list(result)

for i in reversed(result):

seq.append((1 << (n - 1)) | i)

return seq

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generator=2

solution = Solution()

print("输入：", generator)

print("输出：", solution. grayCode(generator))

【例127】迷你Cassandra

3.代码实现

#参数raw\_key是一个字符串，用于哈希

#参数column\_key是一个整数，支持范围查询

#参数column\_value是储存字符串的值

#参数column\_start是开始位置

#参数column\_start是结束位置

#返回值rt是一个由Column对象组成的列表

class Column:

def \_\_init\_\_(self, key, value):

self.key = key

self.value = value

from collections import OrderedDict

class Solution:

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

self.hash = {}

def insert(self, raw\_key, column\_key, column\_value):

if raw\_key not in self.hash:

self.hash[raw\_key] = OrderedDict()

self.hash[raw\_key][column\_key] = column\_value

def query(self, raw\_key, column\_start, column\_end):

rt = []

if raw\_key not in self.hash:

return rt

self.hash[raw\_key] = OrderedDict(sorted(self.hash[raw\_key].items()))

for key, value in self.hash[raw\_key].items():

if key >= column\_start and key <= column\_end:

rt.append(Column(key, value))

return rt

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generator = Column(1, "abcd")

generator1 = Column(2, "hijk")

solution = Solution()

solution.insert("google",generator.key,generator.value)

solution.insert("google",generator1.key,generator1.value)

ls = solution.query("google", 0, 2)

print('输入: query("google", 0, 2)')

print("输出: ")

for i in ls:

print(i.key,i.value)

【例128】网络日志

3.代码实现

#参数timestamp是个整数，建立一个时间点

#返回值是个整数，表示最后五分钟时间点的个数

class WebLogger:

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

self.Q = []

def hit(self, timestamp):

self.Q.append(timestamp)

def get\_hit\_count\_in\_last\_5\_minutes(self, timestamp):

if self.Q == []:

return 0

i = 0

n = len(self.Q)

while i < n and self.Q[i] + 300 <= timestamp:

i += 1

self.Q = self.Q[i:]

return len(self.Q)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = WebLogger()

print("输入：hit(1),hit(2) ")

solution.hit(1)

solution.hit(2)

print("输出最后5分钟时间戳个数：")

print(solution.get\_hit\_count\_in\_last\_5\_minutes(3))

print("输入：hit(300) ")

solution.hit(300)

print("输出最后5分钟时间戳个数：")

print(solution.get\_hit\_count\_in\_last\_5\_minutes(300))

print("输出最后5分钟时间戳个数：")

print(solution.get\_hit\_count\_in\_last\_5\_minutes(301))

【例129】栅栏染色

3.代码实现

#参数n是个非负整数，柱子数

#参数n是个非负整数，颜色数

#返回值是个整数，所有的染色方案

class Solution:

def numWays(self, n, k):

dp = [0, k, k \* k]

if n <= 2:

return dp[n]

if k == 1 and n >= 3:

return 0

for i in range(2, n):

dp.append((k - 1) \* (dp[-1] + dp[-2]))

return dp[-1]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution= Solution()

n=3

k=2

print("输入： n=",n ,"k=",k)

print("输出：",solution.numWays(n,k))

【例130】房屋染色

3.代码实现

#参数costs是个nx3矩阵

#返回值是个整数，刷完所有房子最小花费

class Solution:

def minCost(self, costs):

n = len(costs)

if n == 0:

return 0

INF = 0x7fffffff

f = [costs[0], [INF, INF, INF]]

for i in range(1, n):

for j in range(3):

f[i&1][j] = INF

for k in range(3):

if j != k:

f[i&1][j] = min(f[i&1][j], f[(i+1)&1][k] + costs[i][j])

return min(f[(n-1)&1])

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generator=[[14,2,11],[11,14,5],[14,3,10]]

solution= Solution()

print("输入： ",generator)

print("输出： ",solution.minCost(generator))

【例131】去除重复元素

3.代码实现

#参数nums是个一个整型数组

#返回值result是不重复元素的个数

class Solution:

def deduplication(self, nums):

n = len(nums)

if n == 0:

return 0

nums.sort()

result = 1

for i in range(1, n):

if nums[i - 1] != nums[i]:

nums[result] = nums[i]

result += 1

return result

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generator=[1,3,1,4,4,2]

solution= Solution()

print("输入：",generator)

print("输出：",solution.deduplication(generator))

【例132】左填充

3.代码实现

#参数originalStr是需要添加的字符串

#参数size是目标长度

#参数padchar是在字符串左边填充的字符

#返回值是左填充后的字符串

class StringUtils:

def leftPad(self, originalStr, size, padChar=' '):

return padChar \* (size - len(originalStr)) + originalStr

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

size=8

generator = "foobar"

solution= StringUtils()

print("输入:",generator)

print("输出:",solution.leftPad(generator,size))

【例133】负载均衡器

3.代码实现

#参数server\_id是一个服务器的ID

#返回值是一个ID，集群中随机的一个服务器ID

class LoadBalancer:

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

self.server\_ids = []

self.id2index = {}

def add(self, server\_id):

if server\_id in self.id2index:

return

self.server\_ids.append(server\_id)

self.id2index[server\_id] = len(self.server\_ids) - 1

def remove(self, server\_id):

if server\_id not in self.id2index:

return

# remove the server\_id

index = self.id2index[server\_id]

del self.id2index[server\_id]

# overwrite the one to be removed

last\_server\_id = self.server\_ids[-1]

self.id2index[last\_server\_id] = index

self.server\_ids[index] = last\_server\_id

self.server\_ids.pop()

def pick(self):

import random

index = random.randint(0, len(self.server\_ids) - 1)

return self.server\_ids[index]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution= LoadBalancer()

solution.add(1)

solution.add(2)

solution.remove(1)

print("输入: \nadd(1)\nadd(2)\nremove(1)")

print("输出:",solution.pick())

print("输出:",solution.pick())

【例134】两数和的最接近值

3.代码实现

#参数nums是个整数数组

#参数target是一个整数

#返回值diff是target和两数求和的差距

import sys

class Solution:

def twoSumClosest(self, nums, target):

nums.sort()

i, j = 0, len(nums) - 1

diff = sys.maxsize

while i < j:

if nums[i] + nums[j] < target:

diff = min(diff, target - nums[i] - nums[j])

i += 1

else:

diff = min(diff, nums[i] + nums[j] - target)

j -= 1

return diff

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generator = [-1,2,-1,4]

solution= Solution()

target = 4

print("target =",target)

print("输入：",generator)

print("输出:",solution.twoSumClosest(generator,target))

【例135】打劫房屋

3.代码实现

#参数nums是个非负整数列表，表示每个房子中存放的钱

#返回值是个整数，表示可以拿到的钱

class Solution:

def houseRobber2(self, nums):

n = len(nums)

if n == 0:

return 0

if n == 1:

return nums[0]

dp = [0] \* n

dp[0], dp[1] = 0, nums[1]

for i in range(2, n):

dp[i] = max(dp[i - 2] + nums[i], dp[i - 1])

answer = dp[n - 1]

dp[0], dp[1] = nums[0], max(nums[0], nums[1])

for i in range(2, n - 1):

dp[i] = max(dp[i - 2] + nums[i], dp[i - 1])

return max(dp[n - 2], answer)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generator = [2,3,2,3]

solution= Solution()

print("输入:",generator)

print("输出:",solution.houseRobber2(generator))

【例136】左旋右旋迭代器

3.代码实现

#参数v1，v2表示两个一维向量

#返回值是一个一维数组，交替返回v1，v2元素

class ZigzagIterator:

def \_\_init\_\_(self, v1, v2):

self.queue = [v for v in (v1, v2) if v]

def next(self):

v = self.queue.pop(0)

value = v.pop(0)

if v:

self.queue.append(v)

return value

def hasNext(self):

return len(self.queue) > 0

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

v1= [1,2]

v2= [3,4,5,6]

print("输入:")

print(",".join(str(i) for i in v1))

print(",".join(str(i) for i in v2))

solution, result = ZigzagIterator(v1, v2), []

while solution.hasNext():

result.append(solution.next())

print("输出:",result)

【例137】N数组第K大元素

3.代码实现

import heapq

#参数arrays一个数组列表

#参数k表示第k大

#返回值num是列表中最大的数

class Solution:

def KthInArrays(self, arrays, k):

if not arrays:

return None

# in order to avoid directly changing the original arrays

# and remove the empty arrays, we need a new sortedArrays

sortedArrays = []

for arr in arrays:

if not arr:

continue

sortedArrays.append(sorted(arr, reverse=True))

maxheap = [

(-arr[0], index, 0)

for index, arr in enumerate(sortedArrays)

]

heapq.heapify(maxheap)

num = None

for \_ in range(k):

num, x, y = heapq.heappop(maxheap)

num = -num

if y + 1 < len(sortedArrays[x]):

heapq.heappush(maxheap, (-sortedArrays[x][y + 1], x, y + 1))

return num

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generator = [[2,3,2,4],[3,4,7,9]]

k=5

solution= Solution()

print("输入:",generator)

print("k= ",k)

print("输出:",solution.KthInArrays(generator,k))

【例138】前K大数

3.代码实现

import heapq

#参数nums是个整数数组

#参数k表示第k大

#返回值是个整型数组，前k大的整数组成

class Solution:

def topk(self, nums, k):

heapq.heapify(nums)

topk = heapq.nlargest(k, nums)

topk.sort()

topk.reverse()

return topk

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generator = [8, 7, 6, 5, 4, 3, 2, 1]

k=4

solution= Solution()

print("输入:",generator)

print("k=",k)

print("输出:",solution.topk(generator,k))

【例139】计数型布隆过滤器

3.代码实现

import random

#参数str是个字符串，表示一个word

#返回值是个布尔值，若该word存在返回True，否则返回False

class HashFunction:

def \_\_init\_\_(self, cap, seed):

self.cap = cap

self.seed = seed

def hash(self, value):

ret = 0

for i in value:

ret += self.seed \* ret + ord(i)

ret %= self.cap

return ret

class CountingBloomFilter:

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

self.hashFunc = []

for i in range(k):

self.hashFunc.append(HashFunction(random.randint(10000, 20000), i \* 2 + 3))

self.bits = [0 for i in range(20000)]

def add(self, word):

for f in self.hashFunc:

position = f.hash(word)

self.bits[position] += 1

def remove(self, word):

for f in self.hashFunc:

position = f.hash(word)

self.bits[position] -= 1

def contains(self, word):

for f in self.hashFunc:

position = f.hash(word)

if self.bits[position] <= 0:

return False

return True

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution= CountingBloomFilter(3)

solution.add("long")

solution.add("term")

print('输入:')

print('add("long")')

print('add("term")')

print('contains("long")')

print("输出:",solution.contains("long"))

solution.remove("long")

print('remove("long")')

print('contains("long")')

print("输出:",solution.contains("long"))

【例140】字符计数

3.代码实现

#参数str一个任意的字符串

#返回值map是个哈希map

class Solution:

def countCharacters(self, str):

map = dict()

for c in str:

map[c] = map.get(c, 0) + 1

return map

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generator="abca"

solution = Solution()

print('输入:',generator)

print("输出:",solution.countCharacters(generator))

【例141】最长重复子序列

3.代码实现

#参数str是个任意字符串

#返回值是个整数表示这个字符串最长重复的子序列长度

class Solution:

def longestRepeatingSubsequence(self, str):

n = len(str)

dp = [[0 for j in range(n + 1)] for i in range(n + 1)]

for i in range(1, n + 1):

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

if str[i - 1] == str[j - 1] and i != j:

dp[i][j] = dp[i - 1][j - 1] + 1

else:

dp[i][j] = max(dp[i][j - 1], dp[i - 1][j])

return dp[n][n]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

generator="abcaa"

print('输入：',generator)

print("输出：",solution. longestRepeatingSubsequence(generator))

【例142】僵尸矩阵

3.代码实现

import collections

#参数grid是一个二维整数矩阵

#返回值是个整数，表示需要的天数，若不能完成则返回-1

class Solution:

def zombie(self, grid):

if len(grid) == 0 or len(grid[0]) == 0:

return 0

m, n = len(grid), len(grid[0])

queue = collections.deque()

for i in range(m):

for j in range(n):

if grid[i][j] == 1:

queue.append((i, j))

day = 0

while queue:

size = len(queue)

day += 1

for k in range(size):

(i, j) = queue.popleft()

DIR = [(1, 0), (-1, 0), (0, 1), (0, -1)]

for (di, dj) in DIR:

next\_i, next\_j = i + di, j + dj

if next\_i < 0 or next\_i >= m or next\_j < 0 or next\_j >= n:

continue

if grid[next\_i][next\_j] == 1 or grid[next\_i][next\_j] == 2:

continue

grid[next\_i][next\_j] = 1

queue.append((next\_i, next\_j))

for i in range(m):

for j in range(n):

if grid[i][j] == 0:

return -1

return day - 1

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

generator=[[0,0,0],

[0,0,0],

[0,0,1]]

print("输入：",generator)

print("输出：",solution. zombie(generator))

【例143】摊平二维向量

3.代码实现

class Vector2D(object):

def \_\_init\_\_(self, vec2d):

self.vec2d = vec2d

self.row, self.col = 0, -1

self.next\_elem = None

def next(self):

if self.next\_elem is None:

self.hasNext()

temp, self.next\_elem = self.next\_elem, None

return temp

def hasNext(self):

if self.next\_elem:

return True

self.col += 1

while self.row < len(self.vec2d)and self.col>= len(self.vec2d[self.row]):

self.row += 1

self.col = 0

if self.row < len(self.vec2d) and self.col < len(self.vec2d[self.row]):

self.next\_elem = self.vec2d[self.row][self.col]

return True

return False

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

inputnum=[[1,2],[3],[4,5,6]]

vector2d=Vector2D(inputnum)

print("输入：",inputnum)

print("输出：")

print(vector2d.next())

while vector2d.hasNext():

print(vector2d.next())

【例144】第K大的元素

3.代码实现

class Solution:

# nums是整型数组

# k是整数

# 返回数组第k大的元素

def kthLargestElement2(self, nums, k):

import heapq

heap = []

for num in nums:

heapq.heappush(heap, num)

if len(heap) > k:

heapq.heappop(heap)

return heapq.heappop(heap)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

inputnum=[9,3,2,4,8]

k=3

print("输入数组：",inputnum)

print("输入k=",k)

solution=Solution()

print("输出：",solution.kthLargestElement2(inputnum,k))

【例145】两数和小于或等于目标值

3.代码实现

class Solution:

# 参数nums是整数数组

# 参数target是整数

# 返回整数

def twoSum5(self, nums, target):

l, r = 0, len(nums)-1

cnt = 0

nums.sort()

while l < r:

value = nums[l] + nums[r]

if value > target:

r -= 1

else:

cnt += r - l

l += 1

return cnt

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

inputnum= [2, 7, 11, 15]

target=24

solution=Solution()

print("输入数组：",inputnum)

print("输入target：",target)

solution=Solution()

print("输出：",solution.twoSum5(inputnum,target))

【例146】两数差等于目标值

3.代码实现

class Solution:

#参数nums是整数数组

#参数target是整数

#返回数组的索引值加1，[index1 + 1, index2 + 1] (index1 < index2)

def twoSub(self, nums, target):

nums = [(num, i) for i, num in enumerate(nums)]

target = abs(target)

n, indexs = len(nums), []

nums = sorted(nums, key=lambda x: x[0])

j = 0

for i in range(n):

if i == j:

j += 1

while j < n and nums[j][0] - nums[i][0] < target:

j += 1

if j < n and nums[j][0] - nums[i][0] == target:

indexs = [nums[i][1] + 1, nums[j][1] + 1]

if indexs[0] > indexs[1]:

indexs[0], indexs[1] = indexs[1], indexs[0]

return indexs

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

inputnum= [2, 7, 15, 24]

target=5

solution=Solution()

print("输入数组：",inputnum)

print("输入target：",target)

print("输出：",solution.twoSub(inputnum,target))

【例147】骑士的最短路线

3.代码实现

import collections

class Point:

def \_\_init\_\_(self, a=0, b=0):

self.x = a

self.y = b

DIRECTIONS = [

(-2, -1), (-2, 1), (-1, 2), (1, 2),

(2, 1), (2, -1), (1, -2), (-1, -2),

]

class Solution:

#参数grid表示棋盘

#参数source表示起点

#参数destination表示终点

#返回最短路径长度

def shortestPath(self, grid, source, destination):

queue = collections.deque([(source.x, source.y)])

distance = {(source.x, source.y): 0}

while queue:

x, y = queue.popleft()

if (x, y) == (destination.x, destination.y):

return distance[(x, y)]

for dx, dy in DIRECTIONS:

next\_x, next\_y = x + dx, y + dy

if (next\_x, next\_y) in distance:

continue

if not self.is\_valid(next\_x, next\_y, grid):

continue

distance[(next\_x, next\_y)] = distance[(x, y)] + 1

queue.append((next\_x, next\_y))

return -1

def is\_valid(self, x, y, grid):

n, m = len(grid), len(grid[0])

if x < 0 or x >= n or y < 0 or y >= m:

return False

return not grid[x][y]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

inputnum=[[0,0,0],

[0,0,0],

[0,0,0]]

source = Point(2,0)

destination = Point(2,2)

solution=Solution()

print("输入棋盘：",inputnum)

print("输入起点：[2,0]")

print("输入终点：[2,2]")

print("输出步数：",solution.shortestPath(inputnum,source,destination))

【例148】K个最近的点

3.代码实现

import heapq

import numpy as np

np.set\_printoptions(threshold=np.inf)

class Point:

def \_\_init\_\_(self, a=0, b=0):

self.x = a

self.y = b

class Solution:

#参数points为坐标点列表

#参数origin为初始点

#参数k为整数

#返回k个最邻近点

def kClosest(self, points, origin, k):

self.heap = []

for point in points:

dist = self.getDistance(point, origin)

heapq.heappush(self.heap, (-dist, -point.x, -point.y))

if len(self.heap) > k:

heapq.heappop(self.heap)

ret = []

while len(self.heap) > 0:

\_, x, y = heapq.heappop(self.heap)

ret.append(Point(-x, -y))

ret.reverse()

return ret

def getDistance(self, a, b):

return (a.x - b.x) \*\* 2 + (a.y - b.y) \*\* 2

#主函数

if \_\_name\_\_=='\_\_main\_\_':

a1=Point(0,0)

a2=Point(0,9)

inputnum=[a1,a2]

origin=Point(0,0)

k=1

solution=Solution()

rp=Point(0,0)

rp=solution.kClosest(inputnum,origin,k)

array=[[rp[0].x,rp[0].y]]

print("输入坐标点：[[0,0],[0,9]]")

print("最近坐标数：k=1")

print("输出坐标点：",array)

【例149】优秀成绩

3.代码实现

class Record:

def \_\_init\_\_(self, id, score):

self.id = id

self.score = score

class Solution:

# @param {Record[]} results a list of <student\_id, score>

# @return {dict(id, average)} find the average of 5 highest scores for each person

# <key, value> (student\_id, average\_score)

def highFive(self, results):

# Write your code here

hash = dict()

for r in results:

if r.id not in hash:

hash[r.id] = []

hash[r.id].append(r.score)

if len(hash[r.id]) > 5:

index = 0

for i in range(1, 6):

if hash[r.id][i] < hash[r.id][index]:

index = i

hash[r.id].pop(index)

answer = dict()

for id, scores in hash.items():

answer[id] = sum(scores) / 5.0

return answer

#主函数

if \_\_name\_\_=='\_\_main\_\_':

r1=Record(1,90)

r2=Record(1,93)

r3=Record(2,93)

r4=Record(2,99)

r5=Record(2,98)

r6=Record(2,97)

r7=Record(1,62)

r8=Record(1,56)

r9=Record(2,95)

r10=Record(1,61)

list=[r1,r2,r3,r4,r5,r6,r7,r8,r9,r10]

solution=Solution()

print(solution.highFive(list))

【例150】二叉树的最长连续子序列I

3.代码实现

class TreeNode(object):

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

self.val = x

self.left = None

self.right = None

class Solution:

#参数root是二叉树的根节点

#返回最长连续序列路径的长度

def longestConsecutive2(self, root):

max\_len, \_, \_, = self.helper(root)

return max\_len

def helper(self, root):

if root is None:

return 0, 0, 0

left\_len, left\_down, left\_up = self.helper(root.left)

right\_len, right\_down, right\_up = self.helper(root.right)

down, up = 0, 0

if root.left is not None and root.left.val + 1 == root.val:

down = max(down, left\_down + 1)

if root.left is not None and root.left.val - 1 == root.val:

up = max(up, left\_up + 1)

if root.right is not None and root.right.val + 1 == root.val:

down = max(down, right\_down + 1)

if root.right is not None and root.right.val - 1 == root.val:

up = max(up, right\_up + 1)

len = down + 1 + up

len = max(len, left\_len, right\_len)

return len, down, up

#主函数

if \_\_name\_\_=='\_\_main\_\_':

inputnum={1,2,0,3}

root0=TreeNode(0)

root1=TreeNode(1)

root2=TreeNode(2)

root3=TreeNode(3)

root1.left=root2

root1.right=root0

root2.left=root3

solution=Solution()

print("输入：",inputnum)

print("输出：",solution.longestConsecutive2(root1))

【例151】二叉树的最长连续子序列II

3.代码实现

# 定义一个多节点的树

class MultiTreeNode(object):

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

self.val = x

self.children = [] # children 是 MultiTreeNode 的list

class Solution:

# 参数root为k叉树

# 返回最长连续序列路径的长度

def longestConsecutive3(self, root):

max\_len, \_, \_, = self.helper(root)

return max\_len

def helper(self, root):

if root is None:

return 0, 0, 0

max\_len, up, down = 0, 0, 0

for child in root.children:

result = self.helper(child)

max\_len = max(max\_len, result[0])

if child.val + 1 == root.val:

down = max(down, result[1] + 1)

if child.val - 1 == root.val:

up = max(up, result[2] + 1)

max\_len = max(down + 1 + up, max\_len)

return max\_len, down, up

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

root=MultiTreeNode(5)

root1=MultiTreeNode(6)

root2=MultiTreeNode(4)

root3=MultiTreeNode(7)

root4=MultiTreeNode(5)

root5=MultiTreeNode(8)

root6=MultiTreeNode(3)

root7=MultiTreeNode(5)

root8=MultiTreeNode(3)

root.children=[root1,root2]

root1.children=[root3,root4,root5]

root2.children=[root6,root7,root8]

solution=Solution()

print("输入为：5<6<7<>,5<>,8<>>,4<3<>,5<>,31<>>>")

print("输出为：",solution.longestConsecutive3(root))

【例152】课程表

3.代码实现

from collections import deque

class Solution:

#参数numCourses为整数

#参数prerequisites为先修课列表对

#返回是否能够完成所有课程

def canFinish(self, numCourses, prerequisites):

edges = {i: [] for i in range(numCourses)}

degrees = [0 for i in range(numCourses)]

for i, j in prerequisites:

edges[j].append(i)

degrees[i] += 1

queue, count = deque([]), 0

for i in range(numCourses):

if degrees[i] == 0:

queue.append(i)

while queue:

node = queue.popleft()

count += 1

for x in edges[node]:

degrees[x] -= 1

if degrees[x] == 0:

queue.append(x)

return count == numCourses

#主函数

if \_\_name\_\_=='\_\_main\_\_':

list1=[[1,0]]

n=2

solution=Solution()

print("输入课程数：",n)

print("课程关系为：",list1)

print("输出为：",solution.canFinish(n,list1))

【例153】安排课程

3.代码实现

from queue import Queue

class Solution:

# 参数numCourses为整数

# 参数prerequisites为课程约束关系

# 返回课程顺序

def findOrder(self, numCourses, prerequisites):

edges = {i: [] for i in range(numCourses)}

degrees = [0 for i in range(numCourses)]

for i, j in prerequisites:

edges[j].append(i)

degrees[i] += 1

queue = Queue(maxsize = numCourses)

for i in range(numCourses):

if degrees[i] == 0:

queue.put(i)

order = []

while not queue.empty():

node = queue.get()

order.append(node)

for x in edges[node]:

degrees[x] -= 1

if degrees[x] == 0:

queue.put(x)

if len(order) == numCourses:

return order

return []

#主函数

if \_\_name\_\_ =='\_\_main\_\_':

n=4

list=[[1,0],[2,0],[3,1],[3,2]]

solution=Solution()

print("输入课程数：",n)

print("输入约束为：",list1)

print("输出课程为：",solution.findOrder(n,list1))

【例 154】单词表示数字

3.代码实现

class Solution:

"""

参数number为整数

返回字符串

"""

def convertWords(self, number):

n1 = ["", "one", "two", "three", "four", "five",

"six", "seven", "eight", "nine", "ten",

"eleven", "twelve", "thirteen", "fourteen", "fifteen",

"sixteen", "seventeen", "eighteen", "nineteen"]

n2 = ["", "ten", "twenty", "thirty", "forty",

"fifty", "sixty", "seventy", "eighty", "ninety"]

n3 = ['hundred', '', 'thousand', 'million', 'billion']

res = ''

index = 1

if number == 0:

return 'zero'

elif 0 < number < 20:

return n1[number]

elif 20 <= number < 100:

return n2[number // 10] + ' ' + n1[number]

else:

while number != '':

digit = int(str(number)[-3::])

number = (str(number)[:-3:])

i = len(str(digit))

r = ''

while True:

if digit < 20:

r += n1[digit]

break

elif 20 <= digit < 100:

r += n2[digit // 10] + ' '

elif 100 <= digit < 1000:

r += n1[digit // 100] + ' ' + n3[0] + ' '

digit = digit % (10 \*\* (i - 1))

i -= 1

if digit != 0:

r += ' ' + n3[index] + ' '

index += 1

r += res

res = r

return res.strip()

if \_\_name\_\_=='\_\_main\_\_':

solution=Solution()

n=10245

print("输入为：",n)

print("输出为：",solution.convertWords(n))

【例155】最大子序列的和

3.代码实现

class Solution:

# 参数nums为整型数组

# 参数k为整数

# 返回最大和

def maxSubarray(self, nums, k):

n = len(nums)

if n < k:

return 0

result = 0

for i in range(k):

result += nums[i]

sum = [0 for \_ in range(n + 1)]

min\_prefix = 0

for i in range(1, n + 1):

sum[i] = sum[i - 1] + nums[i - 1]

if i >= k and sum[i] - min\_prefix > result:

result = max(result, sum[i] - min\_prefix)

if i >= k:

min\_prefix = min(min\_prefix, sum[i - k + 1])

return result

#主函数

if \_\_name\_\_=='\_\_main\_\_':

inputnum=[-2,2,-3,4,-1,2,1,-5,3]

k=5

solution=Solution()

print("输入数组为：",inputnum)

print("输入k=：",k)

print("输出sum=：",solution.maxSubarray(inputnum,k))

【例156】移除子串

3.代码实现

class Solution:

# 参数s为字符串

# 参数dict为一组子字符串

# 返回最小长度

def minLength(self, s, dict):

import queue

que = queue.Queue()

que.put(s)

hash = set([s])

min = len(s)

while not que.empty():

s = que.get()

for sub in dict:

found = s.find(sub)

while found != -1:

new\_s = s[:found] + s[found + len(sub):]

if new\_s not in hash:

if len(new\_s) < min:

min = len(new\_s)

que.put(new\_s)

hash.add(new\_s)

found = s.find(sub, found + 1)

return min

#主函数

if \_\_name\_\_=='\_\_main\_\_':

inputwords="ccdaabcdbb"

k=["ab","cd"]

solution=Solution()

print("输入字符串为：",inputwords)

print("输入的子串为：",k)

print("字符串长度为：",solution.minLength(inputwords,k))

【例157】数组划分

3.代码实现

class Solution:

# 参数nums为整型数组

# 参数low为整型

# 参数high整型

# 返回任意可能的解

def partition2(self, nums, low, high):

if len(nums) <= 1:

return

pl, pr = 0, len(nums) - 1

i = 0

while i <= pr:

if nums[i] < low:

nums[pl], nums[i] = nums[i], nums[pl]

pl += 1

i += 1

elif nums[i] > high:

nums[pr], nums[i] = nums[i], nums[pr]

pr -= 1

else:

i += 1

return nums

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

inputnum=[4,3,4,1,2,3,1,2]

low=2

high=3

solution=Solution()

print("输入数组为：",inputnum)

print("输入下限为：",low)

print("输入上限为：",high)

print("输出结果为：",solution.partition2(inputnum,low,high))

【例158】矩形重叠

3.代码实现

# 定义一个点

class Point:

def \_\_init\_\_(self, a=0, b=0):

self.x = a

self.y = b

class Solution:

# 参数l1 第一个长方形左上角的坐标

# 参数r1 第一个长方形右下角的坐标

# 参数l2 第二个长方形左上角的坐标

# 参数r2 第二个长方形右下角的坐标

# 如果重叠返回True

def doOverlap(self, l1, r1, l2, r2):

if l1.x > r2.x or l2.x > r1.x:

return False

if l1.y < r2.y or l2.y < r1.y:

return False

return True

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

l1=Point(0,8)

r1=Point(8,0)

l2=Point(6,6)

r2=Point(10,0)

solution=Solution()

print("输入矩形一：l1=(0,8)，r1=Point(8,0)")

print("输入矩形二：l2=(6,6)，r2=Point(10,0)")

print("输出的结果：",solution.doOverlap(l1,r1,l2,r2))

【例159】最长回文串

3.代码实现

class Solution:

# 参数s 是一个包含大小写的字符串

# 返回能构建的最长回文串4

def longestPalindrome(self, s):

hash = {}

for c in s:

if c in hash:

del hash[c]

else:

hash[c] = True

remove = len(hash)

if remove > 0:

remove -= 1

return len(s) - remove

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

inputnum="abccccdd"

solution=Solution()

print("输入字符串为：",inputnum)

print("输出回文长度：",solution.longestPalindrome(inputnum))

【例160】最大子树

3.代码实现

# 定义一个多节点的树

class TreeNode(object):

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

self.val = x

self.left = None

self.right=None

class Solution:

# 参数root为二叉树根

# 返回最大节点值

import sys

maximum\_weight = 0

result = None

def findSubtree(self, root):

self.helper(root)

return self.result.val

def helper(self, root):

if root is None:

return 0

left\_weight = self.helper(root.left)

right\_weight = self.helper(root.right)

if left\_weight + right\_weight + root.val >= self.maximum\_weight or self.result is None:

self.maximum\_weight = left\_weight + right\_weight + root.val

self.result = root

return left\_weight + right\_weight + root.val

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

root = TreeNode(1)

root1 = TreeNode(-5)

root2 = TreeNode(2)

root3 = TreeNode(0)

root4 = TreeNode(3)

root5 = TreeNode(-4)

root6 = TreeNode(-5)

root.left=root1

root.right=root2

root1.left=root3

root1.right=root4

root2.left=root5

root2.right=root6

solution = Solution()

print("输入为：[1,-5 2,0 3 -4 -5]")

print("输出为：",solution.findSubtree(root))

【例161】最小生成树

3.代码实现

#定义Connection

class Connection:

def \_\_init\_\_(self, city1, city2, cost):

self.city1, self.city2, self.cost = city1, city2, cost

def comp(a, b):

if a.cost != b.cost:

return a.cost - b.cost

if a.city1 != b.city1:

if a.city1 < b.city1:

return -1

else:

return 1

if a.city2 == b.city2:

return 0

elif a.city2 < b.city2:

return -1

else:

return 1

class Solution:

# @param {Connection[]} connections 城市和花费的List

# @return {Connection[]} 返回这个type

def lowestCost(self, connections):

# Write your code here

cmp=0

# connections.sort()

hash = {}

n = 0

for connection in connections:

if connection.city1 not in hash:

n += 1

hash[connection.city1] = n

if connection.city2 not in hash:

n += 1

hash[connection.city2] = n

father = [0 for \_ in range(n + 1)]

results = []

for connection in connections:

num1 = hash[connection.city1]

num2 = hash[connection.city2]

root1 = self.find(num1, father)

root2 = self.find(num2, father)

if root1 != root2:

father[root1] = root2

results.append(connection)

if len(results)!= n - 1:

return []

return results

def find(self, num, father):

if father[num] == 0:

return num

father[num] = self.find(father[num], father)

return father[num]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

conn=Connection("Acity","Bcity",1)

conn1=Connection("Acity","Ccity",2)

conn2=Connection("Bcity","Ccity",3)

connections=[conn,conn1,conn2]

solution = Solution()

ci01=solution.lowestCost(connections)[0].city1

ci02=solution.lowestCost(connections)[0].city2

co0=solution.lowestCost(connections)[0].cost

ci11=solution.lowestCost(connections)[1].city1

ci12=solution.lowestCost(connections)[1].city2

ci1=solution.lowestCost(connections)[1].cost

print([[ci01,ci02,co0],[ci11,ci12,ci1]])

【例162】骑士的最短路径

3.代码实现

import sys

class Solution:

# 参数grid是棋盘

# 返回最短路径长度

def shortestPath2(self, grid):

n = len(grid)

if n == 0:

return -1

m = len(grid[0])

if m == 0:

return -1

f = [ [sys.maxsize for j in range(m)] for \_ in range(n)]

f[0][0] = 0

for j in range(m):

for i in range(n):

if not grid[i][j]:

if i >= 1 and j >= 2 and f[i - 1][j - 2] != sys.maxsize:

f[i][j] = min(f[i][j], f[i - 1][j - 2] + 1)

if i + 1 < n and j >= 2 and f[i + 1][j - 2] != sys.maxsize:

f[i][j] = min(f[i][j], f[i + 1][j - 2] + 1)

if i >= 2 and j >= 1 and f[i - 2][j - 1] != sys.maxsize:

f[i][j] = min(f[i][j], f[i - 2][j - 1] + 1)

if i + 2 < n and j >= 1 and f[i + 2][j - 1] != sys.maxsize:

f[i][j] = min(f[i][j], f[i + 2][j - 1] + 1)

if f[n - 1][m - 1] == sys.maxsize:

return -1

return f[n - 1][m - 1]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

inputnum=[[0,0,0,0],[0,0,0,0],[0,0,0,0]]

solution = Solution()

print("输入为：",inputnum)

print("输出为：",solution.shortestPath2(inputnum))

【例163】最大矩阵

3.代码实现

class Solution:

#参数matrix为矩阵

#返回整数

def maxSquare2(self, matrix):

if not matrix or not matrix[0]:

return 0

n, m = len(matrix), len(matrix[0])

f = [[0] \* m, [0] \* m]

up = [[0] \* m, [0] \* m]

for i in range(m):

f[0][i] = matrix[0][i]

up[0][i] = 1 - matrix[0][i]

edge = max(matrix[0])

for i in range(1, n):

f[i % 2][0] = matrix[i][0]

up[i % 2][0] = 0 if matrix[i][0] else up[(i - 1) % 2][0] + 1

left = 1 - matrix[i][0]

for j in range(1, m):

if matrix[i][j]:

f[i%2][j] =min(f[(i-1)%2][j-1],left,up[(i-1)%2][j])+1

up[i % 2][j] = 0

left = 0

else:

f[i % 2][j] = 0

up[i % 2][j] = up[(i - 1) % 2][j] + 1

left += 1

edge = max(edge, max(f[i % 2]))

return edge \* edge

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

inputnum=[[1,0,1,0,0],[1,0,0,1,0],[1,1,0,0,1],[1,0,0,1,0]]

solution = Solution()

print("输入为：", inputnum)

print("输出为：",solution.maxSquare2(inputnum))

【例164】二叉树的最大节点

3.代码实现

class TreeNode(object):

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

self.val = x

self.left = None

self.right = None

class Solution:

# 参数root为二叉树根

# 返回最大节点值

def maxNode(self, root):

if root is None:

return root

left = self.maxNode(root.left)

right = self.maxNode(root.right)

return self.max(root, self.max(left, right))

def max(self, a, b):

if a is None:

return b

if b is None:

return a

if a.val > b.val:

return a

return b

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

root = TreeNode(1)

root1 = TreeNode(-5)

root2 = TreeNode(3)

root3 = TreeNode(1)

root4 = TreeNode(2)

root5 = TreeNode(-4)

root6 = TreeNode(-5)

root.left = root1

root.right= root2

root1.left = root3

root1.right= root4

root2.left = root5

root2.right= root6

solution = Solution()

print("输入为：[1,-5 3,1 2 -4 -5]")

print("输出为：",solution.maxNode(root).val)

【例165】寻找重复的数

3.代码实现

class Solution:

#参数nums为整型数组

#返回重复的数

def findDuplicate(self, nums):

start, end = 1, len(nums) - 1

while start + 1 < end:

mid = (start + end) // 2

if self.smaller\_than\_or\_equal\_to(nums, mid) > mid:

end = mid

else:

start = mid

if self.smaller\_than\_or\_equal\_to(nums, start) > start:

return start

return end

def smaller\_than\_or\_equal\_to(self, nums, val):

count = 0

for num in nums:

if num <= val:

count += 1

return count

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

inputnum= [5,5,4,3,2,1]

solution = Solution()

print("输入为：",inputnum)

print("输出为：",solution.findDuplicate(inputnum))

【例166】拼字游戏

3.代码实现

import collections

class TrieNode(object):

def \_\_init\_\_(self, value=0):

self.value = value

self.isWord = False

self.children = collections.OrderedDict()

@classmethod

def insert(cls, root, word):

p = root

for c in word:

child = p.children.get(c)

if not child:

child = TrieNode(c)

p.children[c] = child

p = child

p.isWord = True

class Solution:

# 参数board为字符列表

# 参数words为字符串列表

# 返回整数

def boggleGame(self, board, words):

self.board = board

self.words = words

self.m = len(board)

self.n = len(board[0])

self.results = []

self.temp = []

self.visited = [[False for \_ in range(self.n)] for \_ in range(self.m)]

self.root = TrieNode()

for word in words:

TrieNode.insert(self.root, word)

self.dfs(0, 0, self.root)

return len(self.results)

def dfs(self, x, y, root):

for i in range(x, self.m):

for j in range(y, self.n):

paths = []

temp = []

self.getAllPaths(i, j, paths, temp, root)

for path in paths:

word = ''

for px, py in path:

word += self.board[px][py]

self.visited[px][py] = True

self.temp.append(word)

if len(self.temp) > len(self.results):

self.results = self.temp[:]

self.dfs(i, j, root)

self.temp.pop()

for px, py in path:

self.visited[px][py] = False

y = 0

def getAllPaths(self, i, j, paths, temp, root):

if i < 0 or i >= self.m or j < 0 or j >= self.n or \

self.board[i][j] not in root.children or \

self.visited[i][j] == True:

return

root = root.children[self.board[i][j]]

if root.isWord:

temp.append((i,j))

paths.append(temp[:])

temp.pop()

return

self.visited[i][j] = True

deltas = [(0,1), (0,-1), (1,0), (-1, 0)]

for dx, dy in deltas:

newx = i + dx

newy = j + dy

temp.append((i,j))

self.getAllPaths(newx, newy, paths, temp, root)

temp.pop()

self.visited[i][j] = False

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

inputnum= [['a', 'b', 'c'],

['d', 'e', 'f'],

['g', 'h', 'i']]

dictw = ["abc", "cfi", "beh", "defi", "gh"]

solution = Solution()

print("输入字符为：",inputnum)

print("输入字典为：",dictw)

print("输出个数为：",solution.boggleGame(inputnum,dictw))

【例167】132模式识别

3.代码实现

class Solution:

# 参数nums为整数列表

# 返回True或者False

def find132pattern(self, nums):

stk = [-sys.maxsize]

for i in range(len(nums)-1, -1, -1):

if nums[i] < stk[-1]:

return True

else:

while stk and nums[i] > stk[-1]:

v = stk.pop()

stk.append(nums[i])

stk.append(v)

return False

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

inputnum=[1, 2, 3, 4]

solution = Solution()

print("输入为：",inputnum)

print("输出为：",solution.find132pattern(inputnum))

【例168】检查缩写字

3.代码实现

class Solution:

#参数word为字符串

#参数abbr为字符串

#返回布尔类型

def validWordAbbreviation(self, word, abbr):

i = 0

j = 0

while i < len(word) and j < len(abbr):

if word[i] == abbr[j]:

i += 1

j += 1

elif abbr[j].isdigit() and abbr[j] != '0':

start = j

while j < len(abbr) and abbr[j].isdigit():

j += 1

i += int(abbr[start : j])

else:

return False

return i == len(word) and j == len(abbr)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = "internationalization"

abbr = "i12iz4n"

solution = Solution()

print("输入为：",s)

print("缩写为：",abbr)

print("输出为：",solution.validWordAbbreviation(s,abbr))

【例169】一次编辑距离

3.代码实现

class Solution:

# 参数s为字符串

# 参数t为字符串

# 返回布尔类型

def isOneEditDistance(self, s, t):

m = len(s)

n = len(t)

if abs(m - n) > 1:

return False

if m > n:

return self.isOneEditDistance(t, s)

for i in range(m):

if s[i] != t[i]:

if m == n:

return s[i + 1:] == t[i + 1:]

return s[i:] == t[i + 1:]

return m != n

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = "aDb"

t = "adb"

solution = Solution()

print("输入字符串s：",s)

print("输入字符串t：",t)

print("输出为：",solution.isOneEditDistance(s,t))

【例170】数据流滑动窗口平均值

3.代码实现

from collections import deque

class MovingAverage(object):

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

self.queue = deque([])

self.size = size

self.sum = 0.0

def next(self, val):

if len(self.queue) == self.size:

self.sum -= self.queue.popleft()

self.sum += val

self.queue.append(val)

return self.sum / len(self.queue)

if \_\_name\_\_ == '\_\_main\_\_':

solution = MovingAverage(3)

print("输入数据流：1,10,3,5")

print("输出流动窗1：",solution.next(1))

print("输出流动窗2：",solution.next(10))

print("输出流动窗3：",solution.next(3))

print("输出流动窗4：",solution.next(5))

【例171】最长绝对文件路径

3.代码实现

import re

import collections

class Solution:

#参数input为抽象的文件系统

#返回最长文件的绝度路径长度

def lengthLongestPath(self, input):

dict = collections.defaultdict(lambda: "")

lines = input.split("\n")

n = len(lines)

result = 0

for i in range(n):

count = lines[i].count("\t")

lines[i] = dict[count - 1] + re.sub("\\t+","/", lines[i])

if "." in lines[i]:

result = max(result, len(lines[i]))

dict[count] = lines[i]

return result

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

inputwords="dir\n\tsubdir1\n\t\tfile1.ext\n\t\tsubsubdir1\n\tsubdir2\n\t\tsubsubdir2\n\t\t\tfile2.ext"

solution=Solution()

print("输入为：",inputwords)

print("输出为：",solution.lengthLongestPath(inputwords))

【例172】识别名人

3.代码实现

#假定0认识1,1不认识0

class Celebrity:

def knows(a,b):

if a==0 and b==1 :

return True

if a==1 and b==0 :

return False

class Solution:

# 参数n为整数

# 返回整数

def findCelebrity(self, n):

celeb = 0

for i in range(1, n):

if Celebrity.knows(celeb, i):

celeb = i

for i in range(n):

if celeb != i and Celebrity.knows(celeb, i):

return -1

if celeb != i and not Celebrity.knows(i, celeb):

return -1

return celeb

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

n=2

solution=Solution()

print("输入为：",n)

print("输出为：",solution.findCelebrity(n))

【例173】第一个独特字符位置

3.代码实现

class Solution:

# 参数s为字符串

# 返回整数

def firstUniqChar(self, s):

alp = {}

for c in s:

if c not in alp:

alp[c] = 1

else:

alp[c] += 1

index = 0

for c in s:

if alp[c] == 1:

return index

index += 1

return -1

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = "lintcode"

solution=Solution()

print("输入为：",s)

print("输出为：",solution.firstUniqChar(s))

【例174】子串字谜

3.代码实现

class Solution:

# 参数s为字符串

# 参数p为字符串

# 返回索引列表

def findAnagrams(self, s, p):

ans = []

sum = [0 for x in range(0,30)]

plength = len(p)

slength = len(s)

for i in range(plength):

sum[ord(p[i]) - ord('a')] += 1

start = 0

end = 0

matched = 0

while end < slength:

if sum[ord(s[end]) - ord('a')] >= 1:

matched += 1

sum[ord(s[end]) - ord('a')] -= 1

end += 1

if matched == plength:

ans.append(start)

if end - start == plength:

if sum[ord(s[start]) - ord('a')] >= 0:

matched -= 1

sum[ord(s[start]) - ord('a')] += 1

start += 1

return ans

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = "cbaebabacd"

p = "abc"

solution = Solution()

print("输入字符串：",s)

print("输入子串为：",p)

print("输出索引为：",solution.findAnagrams(s,p))

【例175】单词缩写集

3.代码实现

class ValidWordAbbr:

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

self.map = {}

for word in dictionary:

abbr = self.word\_to\_abbr(word)

if abbr not in self.map:

self.map[abbr] = set()

self.map[abbr].add(word)

def word\_to\_abbr(self, word):

if len(word) <= 1:

return word

return word[0] + str(len(word[1:-1])) + word[-1]

def isUnique(self, word):

abbr = self.word\_to\_abbr(word)

if abbr not in self.map:

return True

for word\_in\_dict in self.map[abbr]:

if word != word\_in\_dict:

return False

return True

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

dic=[ "deer", "door", "cake", "card" ]

solution = ValidWordAbbr(dic)

print("输入字典为：",dic)

print("输入单词为： dear")

print("输出结果为：",solution.isUnique("dear"))

print("输入单词为： cart")

print("输出结果为：",solution.isUnique("cart"))

【例176】二叉树翻转

3.代码实现

class TreeNode:

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

self.val = val

self.left, self.right = None, None

class Solution:

def upsideDownBinaryTree(self, root):

if root is None:

return None

return self.dfs(root)

def dfs(self, root):

if root.left is None:

return root

newRoot = self.dfs(root.left)

root.left.right = root

root.left.left = root.right

root.left = None

root.right = None

return newRoot

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

root1 = TreeNode(1)

root2 = TreeNode(2)

root3 = TreeNode(3)

root4 = TreeNode(4)

root5 = TreeNode(5)

inputnum=[1,2,3,4,5,"#","#"]

root1.left = root2

root1.right=root3

root2.left=root4

root2.right=root5

solution = Solution()

a=solution.upsideDownBinaryTree(root1)

a0=a.val

a1=a.left.val

a2=a.right.val

a3='#'

a4='#'

a5=a.right.left.val

a6=a.right.right.val

aa=[a0,a1,a2,a3,a4,a5,a6]

print("输入为",inputnum)

print("输出为",aa)

【例177】 二叉树垂直遍历

3.代码实现

import collections

import queue as Queue

class TreeNode:

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

self.val = val

self.left, self.right = None, None

class Solution:

# 参数root为二叉树根

# 返回整型列表

def verticalOrder(self, root):

results = collections.defaultdict(list)

queue = Queue.Queue()

queue.put((root, 0))

while not queue.empty():

node, x = queue.get()

if node:

results[x].append(node.val)

queue.put((node.left, x - 1))

queue.put((node.right, x + 1))

return [results[i] for i in sorted(results)]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

root=TreeNode(3)

root1=TreeNode(9)

root2=TreeNode(20)

root3=TreeNode(15)

root4=TreeNode(7)

root.left=root1

root.right=root2

root2.left=root3

root2.right=root4

solution = Solution()

a=solution.verticalOrder(root)

print("输入为： [3,9,20,#,#,15,7]")

print("输出为：",a)

【例178】因式分解

3.代码实现

class Solution:

#参数n为整数

#返回组合列表

def getFactors(self, n):

result = []

self.helper(result, [], n, 2);

return result

def helper(self, result, item, n, start):

if n <= 1:

if len(item) > 1:

result.append(item[:])

return

import math

for i in range(start, int(math.sqrt(n)) + 1):

if n % i == 0:

item.append(i)

self.helper(result, item, n / i, i)

item.pop()

if n >= start:

item.append(n)

self.helper(result, item, 1, n)

item.pop()

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

inputnum=8

solution = Solution()

print("输入为：",inputnum)

print("输出为：",solution.getFactors(inputnum))

【例179】Insert Delete GetRandom O(1)

3.代码实现

import random

class RandomizedSet(object):

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

self.nums, self.pos = [], {}

# 参数val为整数

# 返回布尔类型

def insert(self, val):

if val not in self.pos:

self.nums.append(val)

self.pos[val] = len(self.nums) - 1

return True

return False

# 参数val为整数

# 返回布尔类型

def remove(self, val):

if val in self.pos:

idx, last = self.pos[val], self.nums[-1]

self.nums[idx], self.pos[last] = last, idx

self.nums.pop()

del self.pos[val]

return True

return False

def getRandom(self):

return self.nums[random.randint(0, len(self.nums) - 1)]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution=RandomizedSet()

print("插入一个元素：1")

print("输出为：",solution.insert(1))

print("移除一个元素：2")

print("输出为：",solution.remove(2))

print("插入一个元素：2")

print("输出为：",solution.insert(2))

print("获取随机元素：1")

print("输出为：",solution.getRandom())

print("移除一个元素：1")

print("输出为：",solution.remove(1))

print("插入一个元素：2")

print("输出为：",solution.insert(2))

【例180】编码和解码字符串

3.代码实现

class Solution:

#参数strs为字符串列表

#返回编码后的字符串列表

# " " -> ": " 分隔不同单词

# ":" -> "::" 区分":"

def encode(self, strs):

encoded = []

for string in strs:

for char in string:

if char == ":":

encoded.append("::")

else:

encoded.append(char)

encoded.append(": ")

return "".join(encoded)

#参数str为字符串

#返回解码字符串列表

def decode(self, str):

res = []

idx = 0

length = len(str)

tmp\_str = []

while idx < length - 1:

if str[idx] == ":":

if str[idx + 1] == ":":

tmp\_str.append(":")

idx += 2

elif str[idx + 1] == " ":

res.append("".join(tmp\_str))

tmp\_str = []

idx += 2

else:

tmp\_str.append(str[idx])

idx += 1

return res

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

inputwords=["lint","code","love","you"]

solution=Solution()

print("输入为：",inputwords)

print("编码为：",solution.encode(inputwords))

print("解码为：",solution.decode(solution.encode(inputwords)))

【例181】猜数游戏

3.代码实现

def guess(mid):

if mid>4:

return -1

if mid <4:

return 1

if mid==4:

return 0

class Solution:

#参数n为整数

#返回所猜的数

def guessNumber(self, n):

l = 1

r = n

while l <= r:

mid = abs(l + (r - l) / 2)

res = guess(mid)

if res == 0:

return mid

if res == -1:

r = mid - 1

if res == 1:

l = mid + 1

return int(mid)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

inputnum=10

selectedNumber=4

solution=Solution()

print("输入总数为：",inputnum)

print("所选的数字：",selectedNumber)

print("所猜的数字：",solution.guessNumber(inputnum))

【例182】数1的个数

3.代码实现

class Solution:

#参数num为非负整数

#返回数组

def countBits(self, num):

f = [0] \* (num + 1)

for i in range(1, num+1):

f[i] = f[i & i-1] + 1

return f

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

inputnum=5

solution = Solution()

print("输入为：",inputnum)

print("输出为：",solution.countBits(inputnum))

【例183】平面范围求和 -不可变矩阵

3.代码实现

class NumMatrix(object):

#参数matrix为矩阵

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

if len(matrix) == 0 or len(matrix[0]) == 0:

return

n = len(matrix)

m = len(matrix[0])

self.dp = [[0] \* (m + 1) for \_ in range(n + 1)]

for r in range(n):

for c in range(m):

self.dp[r+ 1][c + 1] = self.dp[r + 1][c] + self.dp[r][c + 1] + \

matrix[r][c] - self.dp[r][c]

# 参数row1为整数

# 参数col1为整数

# 参数row2为整数

# 参数col2为整数

# 返回整数

def sumRegion(self, row1, col1, row2, col2):

return self.dp[row2 + 1][col2 + 1] - self.dp[row1][col2 + 1] - \

self.dp[row2 + 1][col1] + self.dp[row1][col1]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

inputnum=[[3,0,1,4,2],[5,6,3,2,1],[1,2,0,1,5],[4,1,0,1,7],[1,0,3,0,5]]

solution = NumMatrix(inputnum)

print("输入矩阵为：",inputnum)

print("区域1的和：",solution.sumRegion(2, 1, 4, 3))

print("区域2的和：",solution.sumRegion(1, 1, 2, 2))

print("区域3的和：",solution.sumRegion(1, 2, 2, 4))

【例184】猜数游戏

3.代码实现

class Solution:

# 参数n为整数

# 返回整数

def getMoneyAmount(self, n):

dp = [[0 for \_ in range(n + 1)] for \_\_ in range(n + 1)]

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

for start in range(1, n - len + 2):

import sys

temp = sys.maxsize

for k in range(start + int((len - 1) / 2), start + int(len - 1)):

left, right = dp[start][k - 1], dp[k + 1][start + len - 1]

temp = min(k + max(left, right), temp)

if left > right:

break

dp[start][start + len - 1] = temp

return dp[1][n]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

inputnum=10

solution = Solution()

print("输入为：",inputnum)

print("输出为：",solution.getMoneyAmount(inputnum))

【例185】最长的回文序列

3.代码实现

class Solution:

# 参数s为字符串

# 返回整数

def longestPalindromeSubseq(self, s):

length = len(s)

if length == 0:

return 0

dp = [[0 for \_ in range(length)] for \_\_ in range(length)]

for i in range(length - 1, -1, -1):

dp[i][i] = 1

for j in range(i + 1, length):

if s[i] == s[j]:

dp[i][j] = dp[i + 1][j - 1] + 2

else:

dp[i][j] = max(dp[i + 1][j], dp[i][j - 1])

return dp[0][length - 1]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

inputnum="bbbab"

solution = Solution()

print("输入为：",inputnum)

print("输出为：",solution.longestPalindromeSubseq(inputnum))

【例186】1和0

3.代码实现

class Solution:

#参数strs为字符串数组

#参数m为整数

#参数n为整数

#返回整数

def findMaxForm(self, strs, m, n):

dp = [[0] \* (m + 1) for \_ in range(n + 1)]

for s in strs:

zero = 0

one = 0

for ch in s:

if ch == "1":

one += 1

else:

zero += 1

for i in range(n,one - 1,-1):

for j in range(m,zero - 1,-1):

if dp[i - one][j - zero] + 1 > dp[i][j]:

dp[i][j] = dp[i - one][j - zero] + 1

return dp[-1][-1]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

inputnum=["10", "0001", "111001", "1", "0"]

m=5

n=3

solution = Solution()

print("输入为：",inputnum)

print("输入m ：",m)

print("输入n ：",n)

print("输出 ：",solution.findMaxForm(inputnum,m,n))

【例187】预测能否胜利

3.代码实现

class Solution:

#参数nums为整数数组

#返回布尔类型

def PredictTheWinner(self, nums):

if len(nums) & 1 == 0: return True

dp = [[0] \* len(nums) for \_ in range(len(nums))]

for i, v in enumerate(nums):

dp[i][i] = v

for i in range(1, len(nums)):

for j in range(len(nums) - i):

dp[j][j + i] = max(nums[j] - dp[j + 1][j + i], nums[j + i] - dp[j][j + i - 1])

return dp[0][-1] > 0

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

inputnum=[1, 5, 2]

solution = Solution()

print("输入为：",inputnum)

print("输出为：",solution.PredictTheWinner(inputnum))

【例188】循环单词

3.代码实现

class Solution:

#参数words为单词列表

#返回整数

def countRotateWords(self, words):

dict1 = set()

for w in words:

s = w + w

for i in range(0, len(w)):

tmp = s[i : i +len(w)]

if tmp in dict1:

dict1.remove(tmp)

dict1.add(w)

return len(dict1)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

dict1 = ["picture", "turepic", "icturep", "word", "ordw", "long"]

solution = Solution()

print("输入为：",dict1)

print("输出为：",solution.countRotateWords(dict1))

【例189】最大子数组之和为k

3.代码实现

class Solution:

#参数nums为数组

#参数k为整数

#返回整数

def maxSubArrayLen(self, nums, k):

m = {}

ans = 0

m[k] = 0

n = len(nums)

sum = [0 for i in range(n + 1)]

for i in range(1, n + 1):

sum[i] = sum[i - 1] + nums[i - 1]

if sum[i] in m:

ans = max(ans, i - m[sum[i]])

if sum[i] + k not in m:

m[sum[i] + k] = i

return ans

if \_\_name\_\_ == '\_\_main\_\_':

num = [-2, 7, 3, -4, 1]

k = 5

solution = Solution()

print("输入数组为：",num)

print("输入目标值：",k)

print("输出为：",solution.maxSubArrayLen(num, k))

【例190】等差切片

3.代码实现

class Solution(object):

def numberOfArithmeticSlices(self, A):

#参数A为列表

#返回整数

size = len(A)

if size < 3: return 0

ans = cnt = 0

delta = A[1] - A[0]

for x in range(2, size):

if A[x] - A[x - 1] == delta:

cnt += 1

ans += cnt

else:

delta = A[x] - A[x - 1]

cnt = 0

return ans

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

inputnum=[1, 2, 3, 4]

print("输入为：",inputnum)

print("输出为：",solution.numberOfArithmeticSlices(inputnum))

【例191】2D战舰

3.代码实现

class Solution(object):  
 def countBattleships(self, board):  
 #参数board为列表  
 #返回整数

len1 = len(board)  
 if len1 == 0:  
 return 0;  
 len2 = len(board[0])  
 ans = 0  
 for i in range(0, len1):  
 for j in range(0,len2):  
 if board[i][j] == 'X' and ( i == 0 or board[i-1][j] == '.' )and (j == 0 or board[i][j-1] == '.'):  
 ans += 1  
 return ans  
if \_\_name\_\_ == '\_\_main\_\_':  
 solution=Solution()  
 inputnum=["X..X","...X","...X"]

print("输入为：",inputnum)

print("输出为：",solution.countBattleships(inputnum))

【例192】连续数组

3.代码实现

class Solution:

#参数nums为数组

#返回整数

def findMaxLength(self, nums):

index\_sum = {}

cur\_sum = 0

ans = 0

for i in range(len(nums)):

if nums[i] == 0: cur\_sum -= 1

else: cur\_sum += 1

if cur\_sum == 0: ans = i+1

elif cur\_sum in index\_sum: ans = max(ans, i-index\_sum[cur\_sum])

if cur\_sum not in index\_sum: index\_sum[cur\_sum] = i

return ans

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

inputnum=[1,0]

print("输入为：",inputnum)

print("输出为：",solution.findMaxLength(inputnum))

【例193】带有冷却时间的买卖股票最佳时间

3.代码实现

class Solution:

#参数prices为整数列表

#返回整数

def maxProfit(self, prices):

if not prices:

return 0

buy, sell, cooldown = [0 for \_ in range(len(prices))], [0 for \_ in range(len(prices))], [0 for \_ in range(len(prices))]

buy[0] = -prices[0]

for i in range(1, len(prices)):

cooldown[i] = sell[i - 1]

sell[i] = max(sell[i - 1], buy[i - 1] + prices[i])

buy[i] = max(buy[i - 1], cooldown[i - 1] - prices[i])

return max(sell[-1], cooldown[-1])

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

inputnum=[1,2,3,0,2]

print("输入为：",inputnum)

print("输出为：",solution.maxProfit(inputnum))  
4.运行结果

输入为：[1, 2, 3, 0, 2]

输出为：3

【例194】小行星的碰撞

3.代码实现

class Solution:

#参数asteroids为整数数组

#返回整数数组

def asteroidCollision(self, asteroids):

ans, i, n= [], 0, len(asteroids)

while i < n:

if asteroids[i] > 0:

ans.append(asteroids[i])

elif len(ans) == 0 or ans[-1] < 0:

ans.append(asteroids[i])

elif ans[-1] <= -asteroids[i]:

if ans[-1] < -asteroids[i]:

i -= 1

ans.pop()

i += 1

return ans

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

inputnum=[5,10,-5]

print("输入为：",inputnum)

print("输出为：",solution.asteroidCollision(inputnum))  
4.运行结果  
输入为：[5, 10, -5]

输出为：[5, 10]

【例195】扩展弹性词

3.代码实现

class Solution:

#参数S为字符串

#参数words为字符串列表

#返回整数

def expressiveWords(self, S, words):

SList = self.countGroup(S)

n = len(SList)

ans = 0

for word in words:

wordList = self.countGroup(word)

if n != len(wordList):

continue

ok = 1

for i in range(n):

if not self.canExtend(wordList[i], SList[i]):

ok = 0

break

ans += ok

return ans

def countGroup(self, s):

n = len(s)

cnt = 1

ret = []

for i in range(1, n):

if s[i] == s[i - 1]:

cnt += 1

else:

ret.append((s[i - 1], cnt))

cnt = 1

ret.append((s[-1], cnt))

return ret

def canExtend(self, From, To):

return From[0] == To[0] and \

(From[1] == To[1] or (From[1] < To[1] and To[1] >= 3))

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

inputnum1="heeellooo"

inputnum2=["hello", "hi", "helo"]

print("输入字符串1为：",inputnum1)

print("输入字符串2为：",inputnum2)

print("输出为：",solution.expressiveWords(inputnum1,inputnum2))

【例196】找到最终的安全状态

3.代码实现

class Solution:

#参数graph为整数数组

#返回整数

def eventualSafeNodes(self, graph):

def dfs(graph, i, visited):

for j in graph[i]:

if j in visited:

return False

if j in ans:

continue

visited.add(j)

if not dfs(graph, j, visited):

return False

visited.remove(j)

ans.add(i)

return True

ans = set()

for i in range(len(graph)):

visited = set([i])

dfs(graph, i, visited)

return sorted(list(ans))

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

inputnum=[[1,2],[2,3],[5],[0],[5],[],[]]

print("输入为：",inputnum)

print("输出为：",solution.eventualSafeNodes(inputnum))

【例197】使序列递增的最小交换次数

3.代码实现

class Solution:

def minSwap(self, A, B):

if len(A) == 0 or len(A) != len(B):

return 0

non\_swapped, swapped = [0] \* len(A), [1] + [0] \* (len(A) - 1)

for i in range(1, len(A)):

swps, no\_swps = set(), set()

if A[i - 1] < A[i] and B[i - 1] < B[i]:

swps.add(swapped[i - 1] + 1)

no\_swps.add(non\_swapped[i - 1])

if B[i - 1] < A[i] and A[i - 1] < B[i]:

swps.add(non\_swapped[i - 1] + 1)

no\_swps.add(swapped[i - 1])

swapped[i], non\_swapped[i] = min(swps), min(no\_swps)

return min(swapped[-1], non\_swapped[-1])

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

inputnum1=[1,3,5,4]

inputnum2=[1,2,3,7]

print("输入1为：",inputnum1)

print("输入2为：",inputnum2)

print("输出为：",solution.minSwap(inputnum1,inputnum2))

【例198】所有可能的路径

3.代码实现

class Solution:

#参数graph为数组

#返回数组

def allPathsSourceTarget(self, graph):

N = len(graph)

res = []

def dfs(N, graph, start, res, path):

if start == N-1:

res.append(path)

else:

for node in graph[start]:

dfs(N, graph, node, res, path + [node])

dfs(N, graph, 0, res, [0])

return (res)

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

inputnum=[[1,2],[3],[3],[]]

print("输入为：",inputnum)

print("输出为：",solution.allPathsSourceTarget(inputnum))

【例199】合法的井字棋状态

3.代码实现

class Solution:

def validTicTacToe(self, board):

#参数board为列表

#返回布尔类型

num\_X, num\_O = 0, 0

for i in range(0, 3):

for j in range(0, 3):

if board[i][j] == 'X':

num\_X += 1

if board[i][j] == 'O':

num\_O += 1

if not (num\_X == num\_O or num\_X == num\_O + 1):

return False

for i in range(3):

if board[i][0] == board[i][1] == board[i][2]:

if board[i][0] == 'X':

return num\_X == num\_O + 1

if board[i][0] == 'O':

return num\_X == num\_O

for j in range(3):

if board[0][j] == board[1][j] == board[2][j]:

if board[0][j] == 'X':

return num\_X == num\_O + 1

if board[0][j] == 'O':

return num\_X == num\_O

if board[0][0] == board[1][1] == board[2][2]:

if board[0][0] == 'X':

return num\_X == num\_O + 1

if board[0][0] == 'O':

return num\_X == num\_O

if board[0][2] == board[1][1] == board[2][0]:

if board[2][0] == 'X':

return num\_X == num\_O + 1

if board[2][0] == 'O':

return num\_X == num\_O

return True

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

inputnum=["O ", " ", " "]

print("输入为：",inputnum)

print("输出为：",solution.validTicTacToe(inputnum))

【例200】满足要求的子串个数

3.代码实现

class Solution:

#参数S为字符串

#参数words为单词字典

#返回子串的个数

def numMatchingSubseq(self, S, words):

self.idx = {'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4,

'f': 5, 'g': 6, 'h': 7, 'i': 8, 'j': 9,

'k': 10, 'l': 11, 'm': 12, 'n': 13, 'o': 14,

'p': 15, 'q': 16, 'r': 17, 's': 18, 't': 19,

'u': 20, 'v': 21, 'w': 22, 'x': 23, 'y': 24, 'z': 25}

n = len(S)

nxtPos = []

tmp = [-1] \* 26

for i in range(n - 1, -1, -1):

tmp[self.idx[S[i]]] = i

nxtPos.append([i for i in tmp])

nxtPos = nxtPos[::-1]

ans = 0

for word in words:

if self.isSubseq(word, nxtPos):

ans += 1

return ans

def isSubseq(self, word, nxtPos):

lenw = len(word)

lens = len(nxtPos)

i, j = 0, 0

while i < lenw and j < lens:

j = nxtPos[j][self.idx[word[i]]]

if j < 0:

return False

i += 1

j += 1

return i == lenw

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

input1="abcde"

input2=["a", "bb", "acd", "ace"]

print("输入字符串为：",input1)

print("输入子串为：",input2)

print("输出为：",solution.numMatchingSubseq(input1,input2))

【例201】多米诺和三格骨牌铺瓦问题

3.代码实现

class Solution:

#参数N为整数

#返回整数

def numTilings(self, N):

if N < 3:

return N

MOD = 1000000007

f = [[0, 0, 0] for i in range(N + 1)]

f[0][0] = f[1][0] = f[1][1] = f[1][2] = 1

for i in range(2, N + 1):

f[i][0] = (f[i - 1][0] + f[i - 2][0] + f[i - 2][1] + f[i - 2][2]) % MOD;

f[i][1] = (f[i - 1][0] + f[i - 1][2]) % MOD;

f[i][2] = (f[i - 1][0] + f[i - 1][1]) % MOD;

return f[N][0]

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

inputnum=3

print("输入为：",inputnum)

print("输出为：",solution.numTilings(inputnum))

【例202】逃离幽灵

3.代码实现

class Solution:

#参数ghosts为数组

#参数target为数组

#返回布尔类型

def escapeGhosts(self, ghosts, target):

target\_dist = abs(target[0]) + abs(target[1])

for r, c in ghosts:

ghost\_target = abs(target[0] - r) + abs(target[1] - c)

if ghost\_target <= target\_dist:

return False

return True

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

inputnum1=[[1, 0], [0, 3]]

inputnum2=[0, 1]

print("输入幽灵为：",inputnum1)

print("输入目标为：",inputnum2)

print("输出为：",solution.escapeGhosts(inputnum1,inputnum2))

【例203】寻找最便宜的航行旅途（最多经过k个中转站）

1.问题描述

有n个城市由航班连接，每个航班 (u,v,w)表示从城市u出发，到达城市v，价格为w。给定城市数目n，所有的航班flights。找到从起点src到终点站dst线路最便宜的价格，而旅途中最多只能中转K次。如果没有找到合适的线路，返回-1。

2.问题示例

输入n = 3，flights = [[0,1,100],[1,2,100],[0,2,500]]，src = 0，dst = 2，K = 0，输出500，即不中转的条件下，最便宜的价格为500。

3.代码实现

import sys

class Solution:

#参数n为整数

#参数flights为矩阵

#参数src为整数

#参数dst为整数

#参数K为整数

#返回整数

def findCheapestPrice(self, n, flights, src, dst, K):

distance = [sys.maxsize for i in range(n)]

distance[src] = 0

for i in range(0, K + 1):

dN = list(distance)

for u, v, c in flights:

dN[v] = min(dN[v], distance[u] + c)

distance = dN

if distance[dst] != sys.maxsize:

return distance[dst]

else:

return -1

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

n=3

flights=[[0, 1, 100], [1, 2, 100], [0, 2, 500]]

src=0

dst=2

K=0

print("输入城市为：",n)

print("输入航班为：",flights)

print("输入出发地：",src)

print("输入目的地：",dst)

print("输入中转数：",K)

print("输出价格为：",solution.findCheapestPrice(n, flights, src, dst, K))

4.运行结果

输入城市为：3

输入航班为：[[0, 1, 100], [1, 2, 100], [0, 2, 500]]

输入出发地：0

输入目的地：2

输入中转数：0

输出价格为：500

【例204】图是否可以被二分

3.代码实现

class Solution:

#参数graph为无向图

#返回布尔类型

def isBipartite(self, graph):

n = len(graph)

self.color = [0] \* n

for i in range(n):

if self.color[i] == 0 and not self.colored(i, graph, 1):

return False

return True

def colored(self, now, graph, c):

self.color[now] = c

for nxt in graph[now]:

if self.color[nxt] == 0 and not self.colored(nxt, graph, -c):

return False

elif self.color[nxt] == self.color[now]:

return False

return True

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

inputnum=[[1,3],[0,2],[1,3],[0,2]]

print("输入为：",inputnum)

print("输出为：",solution.isBipartite(inputnum))  
4.运行结果

输入为：[[1, 3], [0, 2], [1, 3], [0, 2]]

输出为：True

【例205】森林中的兔子

3.代码实现

import math

class Solution:

#参数answers为数组

#返回整数

def numRabbits(self, answers):

hsh = {}

for i in answers:

if i + 1 in hsh:

hsh[i + 1] += 1

else:

hsh[i + 1] = 1

ans = 0

for i in hsh:

ans += math.ceil(hsh[i] / i) \* i

return ans

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

inputnum=[1,1,2]

print("输入为：",inputnum)

print("输出为：",solution.numRabbits(inputnum))  
4.运行结果

输入为：[1, 1, 2]

输出为：5

【例206】最大分块排序

3.代码实现

class Solution(object):

def maxChunksToSorted(self, arr):

def dfs(cur, localmax):

visited[cur] = True

localmax = max(localmax, cur)

if not visited[arr[cur]]:

return dfs(arr[cur], localmax)

return localmax

visited = [False] \* len(arr)

count = 0

i = 0

while i < len(arr):

localmax = dfs(i, -1)

i += 1

while i < localmax + 1:

if not visited[i]:

localmax = dfs(i, localmax)

i += 1

count += 1

return count

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

arr = [1,0,2,3,4]

print("输入为：",arr)

print("输出为：",solution.maxChunksToSorted(arr))  
4.运行结果

输入为：[1, 0, 2, 3, 4]

输出为：4

【例207】分割标签

3.代码实现

class Solution(object):

def partitionLabels(self, S):

last = {c: i for i, c in enumerate(S)}

right = left = 0

ans = []

for i, c in enumerate(S):

right = max(right, last[c])

if i == right:

ans.append(i - left + 1)

left = i + 1

return ans

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

s = "ababcbacadefegdehijhklij"

print("输入为：",s)

print("输出为：",solution.partitionLabels(s))  
4.运行结果

输入为：ababcbacadefegdehijhklij

输出为：[9, 7, 8]

【例208】网络延迟时间

3.代码实现

class Solution:

#参数times为数组

#参数N为整数

#参数K为整数

#返回整数

def networkDelayTime(self, times, N, K):

INF = 0x3f3f3f3f

G = [[INF for i in range(N + 1)] for j in range(N + 1)]

for i in range(1, N + 1):

G[i][i] = 0

for i in range(0, len(times)):

G[times[i][0]][times[i][1]] = times[i][2]

dis = G[K][:]

vis = [0] \* (N + 1)

for i in range(0, N - 1):

Mini = INF

p = K

for j in range(1, N + 1):

if vis[j] == 0 and dis[j] < Mini:

Mini = dis[j]

p = j

vis[p] = 1

for j in range(1, N + 1):

if vis[j] == 0 and dis[j] > dis[p] + G[p][j]:

dis[j] = dis[p] + G[p][j]

ans = 0

for i in range(1, N + 1):

ans = max(ans, dis[i])

if ans == INF:

return -1

return ans

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

times=[[2,1,1],[2,3,1],[3,4,1]]

N=4

K=2

print("时间矩阵为：",times)

print("网络大小为：",N)

print("起始节点为：",K)

print("最小花费为：",solution.networkDelayTime(times,N,K))

【例209】洪水填充

3.代码实现

class Solution(object):

def floodFill(self, image, sr, sc, newColor):

rows, cols, orig\_color = len(image), len(image[0]), image[sr][sc]

def traverse(row, col):

if (not (0 <= row < rows and 0 <= col < cols)) or image[row][col] != orig\_color:

return

image[row][col] = newColor

[traverse(row + x, col + y) for (x, y) in ((0, 1), (1, 0), (0, -1), (-1, 0))]

if orig\_color != newColor:

traverse(sr, sc)

return image

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

image = [[1,1,1],[1,1,0],[1,0,1]]

sr = 1

sc = 1

newColor = 2

print("输入图像为：",image)

print("输入坐标为： [",sr,",",sc,"]")

print("输入颜色为：",newColor)

print("输出图像为：",(solution.floodFill(image,sr,sc,newColor)))

【例210】映射配对之和

3.代码实现

class TrieNode:

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

self.son = {}

self.val = 0

class Trie:

root = TrieNode()

def insert(self, s, val):

cur = self.root

for i in range(0, len(s)):

if s[i] not in cur.son:

cur.son[s[i]] = TrieNode()

cur = cur.son[s[i]]

cur.val += val

def find(self, s):

cur = self.root

for i in range(0, len(s)):

if s[i] not in cur.son:

return 0

cur = cur.son[s[i]]

return cur.val

class MapSum:

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

self.d = {}

self.trie = Trie()

def insert(self, key, val):

#参数key为字符串

#参数val为整数

#无返回值

if key in self.d:

self.trie.insert(key, val - self.d[key])

else:

self.trie.insert(key, val)

self.d[key] = val

def sum(self, prefix):

#参数prefix为字符串

#返回整型

return self.trie.find(prefix)

if \_\_name\_\_ == '\_\_main\_\_':

mapsum=MapSum()

print("插入方法：")

print(mapsum.insert("apple", 3))

print("求和方法：")

print(mapsum.sum("ap"))

print("插入方法：")

print(mapsum.insert("app", 2))

print("求和方法：")

print(mapsum.sum("ap"))  
4.运行结果

插入方法：None

求和方法：3

插入方法：None

求和方法：5

【例211】最长升序子序列的个数

3.代码实现

import collections

class Solution(object):

def findNumberOfLIS(self, nums):

ans = [0, 0]

l = len(nums)

dp = collections.defaultdict(list)

for i in range(l):

dp[i] = [1, 1]

for i in range(l):

for j in range(i):

if nums[i] > nums[j]:

if dp[j][0] + 1 > dp[i][0]:

dp[i] = [dp[j][0] + 1, dp[j][1]]

elif dp[j][0] + 1 == dp[i][0]:

dp[i] = [dp[i][0], dp[i][1] + dp[j][1]]

for i in dp.keys():

if dp[i][0] > ans[0]:

ans = [dp[i][0], dp[i][1]]

elif dp[i][0] == ans[0]:

ans = [dp[i][0], ans[1] + dp[i][1]]

return ans[1]

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

nums=[1,3,5,4,7]

print("输入为：",nums)

print("输出为：",solution.findNumberOfLIS(nums))

【例212】最大的交换

3.代码实现

class Solution:

def maximumSwap(self, num):

res, num = num, list(str(num))

for i in range(len(num) - 1):

for j in range(i + 1, len(num)):

if int(num[j]) > int(num[i]):

tmp = int("".join(num[:i] + [num[j]] + num[i + 1:j] + [num[i]] + num[j + 1:]))

res = max(res, tmp)

return res

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

num=2736

print("输入为：",num)

print("输出为：",solution.maximumSwap(num))

【例213】将数组拆分成含有连续元素的子序列

3.代码实现

class Solution:

#参数nums为整数列表

#返回布尔类型

def isPossible(self, nums):

cnt, tail = {}, {}

for num in nums:

cnt[num] = cnt[num] + 1 if num in cnt else 1

for num in nums:

if not num in cnt or cnt[num] < 1:

continue

if num - 1 in tail and tail[num - 1] > 0:

tail[num - 1] -= 1

tail[num] = tail[num] + 1 if num in tail else 1

elif num + 1 in cnt and cnt[num + 1] > 0 and num + 2 in cnt and cnt[num + 2] > 0:

cnt[num + 1] -= 1

cnt[num + 2] -= 1

tail[num + 2] = tail[num + 2] + 1 if num + 2 in tail else 1

else:

return False

cnt[num] -= 1

return True

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

nums=[1,2,3,3,4,5]

print("输入为：",nums)

print("输出为：",solution.isPossible(nums))

【例214】Dota2参议院

3.代码实现

from collections import deque

class Solution():

def predictPartyVictory(self,senate):

senate = deque(senate)

while True:

try:

thisGuy = senate.popleft()

if thisGuy == 'R':

senate.remove('D')

else:

senate.remove('R')

senate.append(thisGuy)

except:

return 'Radiant' if thisGuy == 'R' else 'Dire'

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

senate="RD"

print("输入为：",senate)

print("输出为：",solution.predictPartyVictory(senate))

【例215】合法的三角数

3.代码实现

class Solution:

#参数nums为数组

#返回整数

def triangleNumber(self, nums):

nums = sorted(nums)

total = 0

for i in range(len(nums)-2):

if nums[i] == 0:

continue

end = i + 2

for j in range(i+1, len(nums)-1):

while end < len(nums) and nums[end] < (nums[i] + nums[j]):

end += 1

total += end - j - 1

return total

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

nums=[2,2,3,4]

print("输入为：",nums)

print("输出为：",solution.triangleNumber(nums))

【例216】在系统中找到重复文件

3.代码实现

import collections

class Solution:

def findDuplicate(self, paths):

dic = collections.defaultdict(list)

for path in paths:

root, \*f = path.split(" ")

for file in f:

txt, content = file.split("(")

dic[content] += root + "/" + txt,

return [dic[key] for key in dic if len(dic[key]) > 1]

if \_\_name\_\_ == '\_\_main\_\_':

paths=["root/a 1.txt(abcd) 2.txt(efgh)", "root/c 3.txt(abcd)","root/c/d 4.txt(efgh)"]

solution=Solution()

print("输入为：",paths)

print("输出为：",solution.findDuplicate(paths))

【例217】两个字符串的删除操作

3.代码实现

class Solution:

#word1为字符串

#参数word2为字符串

#返回整数

def minDistance(self, word1, word2):

m, n = len(word1), len(word2)

dp = [[0] \* (n + 1) for i in range(m + 1)]

for i in range(m):

for j in range(n):

dp[i + 1][j + 1] = max(dp[i][j + 1], dp[i + 1][j], dp[i][j] + (word1[i] == word2[j]))

return m + n - 2 \* dp[m][n]

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

word1="sea"

word2="eat"

print("输入1为：",word1)

print("输入2为：",word2)

print("输出为：",solution.minDistance(word1,word2))

【例218】下一个更大的元素

3.代码实现

class Solution:

#n为整数

#返回整数

def nextGreaterElement(self, n):

n\_array = list(map(int, list(str(n))))

if len(n\_array) <= 1:

return -1

if len(n\_array) == 2:

if n\_array[0] < n\_array[1]:

return int("".join(map(str, n\_array[::-1])))

else:

return -1

if n\_array[-2] < n\_array[-1]:

n\_array[-2], n\_array[-1] = n\_array[-1], n\_array[-2]

new\_n = int("".join(map(str, n\_array)))

else:

i = len(n\_array) - 1

while i > 0 and n\_array[i - 1] >= n\_array[i]:

i -= 1

if i == 0:

return -1

else:

new\_array = n\_array[:i - 1]

for j in range(len(n\_array) - 1, i - 1, -1):

if n\_array[j] > n\_array[i - 1]:

break

new\_array.append(n\_array[j])

n\_array[j] = n\_array[i - 1]

new\_array.extend(reversed(n\_array[i:]))

new\_n = int("".join(map(str, new\_array)))

return new\_n if new\_n <= 2 \*\* 31 else -1

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

n=123

print("输入为：",n)

print("输出为：",solution.nextGreaterElement(n))

【例219】最优除法

3.代码实现

class Solution(object):

def optimalDivision(self, nums):

joinDivision = lambda l: '/'.join(list(map(str,l)))

if len(nums) == 1: return str(nums[0])

if len(nums) == 2: return joinDivision(nums)

return str(nums[0]) if len(nums) == 1 else str(nums[0]) + '/(' + joinDivision(nums[1:]) +')'

if \_\_name\_\_ == '\_\_main\_\_':

nums=[1000,100,10,2]

solution=Solution()

print("输入为：",nums)

print("输出为：",solution.optimalDivision(nums))

【例220】通过删除字母匹配到字典里最长单词

3.代码实现

class Solution:

#参数s为字符串

#参数d为列表

#返回字符串

def findLongestWord(self, s, d):

for x in sorted(d, key=lambda x: (-len(x), x)):

it = iter(s)

if all(c in it for c in x):

return x

return ''

if \_\_name\_\_ == '\_\_main\_\_':

s = "abpcplea"

d = ["ale", "apple", "monkey", "plea"]

solution=Solution()

print("输入为：",s)

print("输入为：",d)

print("输出为：",solution.findLongestWord(s,d))

【例221】寻找树中最左下结点的值

3.代码实现

class TreeNode:

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

self.val = val

self.left, self.right = None, None

class Solution:

#参数root为二叉树根

#返回整数

def findBottomLeftValue(self, root):

self.max\_level = 0

self.val = None

self.helper(root, 1)

return self.val

def helper(self, root, level):

if not root: return

if level > self.max\_level:

self.max\_level = level

self.val = root.val

self.helper(root.left, level + 1)

self.helper(root.right, level + 1)

if \_\_name\_\_ == '\_\_main\_\_':

node=TreeNode(1)

node.left=TreeNode(2)

node.right=TreeNode(3)

node.left.left=TreeNode(4)

solution=Solution()

print("输入为：[1,2 3,4 # # #}"]

print("输出为：",solution.findBottomLeftValue(node))

【例222】出现频率最高的子树和

3.代码实现

import collections

class TreeNode:

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

self.val = val

self.left, self.right = None, None

class Solution:

def findFrequentTreeSum(self, root):

#root为树根节点

#返回列表

if not root:

return []

counter = collections.Counter()

def sumnode(node):

if not node:

return 0

ret = node.val

if node.left:

ret += sumnode(node.left)

if node.right:

ret += sumnode(node.right)

counter[ret] += 1

return ret

sumnode(root)

arr = []

for k in counter:

arr.append((k, counter[k]))

arr.sort(key=lambda x: x[1], reverse=True)

i = 0

while i + 1 < len(arr) and arr[i + 1][1] == arr[0][1]:

i += 1

return [x[0] for x in arr[:i + 1]]

if \_\_name\_\_ == '\_\_main\_\_':

node=TreeNode(5)

node.right=TreeNode(-3)

node.left=TreeNode(2)

solution=Solution()

print("输入为：{5，3 2}")

print("输出为：",solution.findFrequentTreeSum(node))

【例223】寻找BST的modes

3.代码实现

class TreeNode:

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

self.val = val

self.left, self.right = None, None

class Solution:

#参数root为根节点

#返回整数

def helper(self, root, cache):

if root == None:

return

cache[root.val] += 1

self.helper(root.left, cache)

self.helper(root.right, cache)

return

def findMode(self, root):

from collections import defaultdict

if root == None:

return []

cache = defaultdict(int)

self.helper(root, cache)

max\_freq = max(cache.values())

result = [k for k,v in cache.items() if v == max\_freq]

return result

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

T = TreeNode(1)  
 T.left = None  
 T2 = TreeNode(2)  
 T.right = T2  
 T3 = TreeNode(2)  
 T2.left = T3  
 s = Solution()  
 print("输入为：[1,#,2,2]")

print("输出为：",s.findMode(T))

【例224】对角线遍历

3.代码实现

class Solution:  
 #参数matrix为矩阵  
 #返回整数列表  
def findDiagonalOrder(self, matrix):  
 import collections  
 result = [ ]  
 dd = collections.defaultdict(list)  
 if not matrix:  
 return result  
 for i in range(0, len(matrix)):  
 for j in range(0, len(matrix[0])):  
 dd[i+j+1].append(matrix[i][j])  
 for k, v in dd.items():  
 if k%2==1: dd[k].reverse()  
 result += dd[k]  
 return result

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

m = [  
[ 1, 2, 3 ],  
[ 4, 5, 6 ],  
[ 7, 8, 9 ]  
]

s = Solution()

print("输入为：",m)

print("输出为：",s.findDiagonalOrder(m))

【例225】提莫攻击

3.代码实现

class Solution:

#参数timeSeries为整数数组

#参数duration为整数

#返回整数

def findPoisonedDuration(self, timeSeries, duration):

ans = duration \* len(timeSeries)

for i in range(1,len(timeSeries)):

ans -= max(0, duration - (timeSeries[i] - timeSeries[i-1]))

return ans

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

time =2

timws = [1,4]

print("输入攻击序列：",timws)

print("输入持续时间：",time)

print("输出中毒时间：",s.findPoisonedDuration(timws,time))

【例226】目标和

3.代码实现

class Solution(object):

def findTargetSumWays(self, nums, S):

if not nums:

return 0

dic = {nums[0]: 1, -nums[0]: 1} if nums[0] != 0 else {0: 2}

for i in range(1, len(nums)):

tdic = {}

for d in dic:

tdic[d + nums[i]] = tdic.get(d + nums[i], 0) + dic.get(d, 0)

tdic[d - nums[i]] = tdic.get(d - nums[i], 0) + dic.get(d, 0)

dic = tdic

return dic.get(S, 0)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

time =3

timws = [1,1,1,1,1]

print("输入目标值：",time)

print("输入序列值：",timws)

print("输出方法为：",s.findTargetSumWays(timws,time))

【例227】升序子序列

3.代码实现

class Solution(object):

def findSubsequences(self, nums):

#参数nums为列表

#返回列表

res = []

self.subsets(nums, 0, [], res)

return res

def subsets(self, nums, index, temp, res):

if len(nums) >= index and len(temp) >= 2:

res.append(temp[:])

used = {}

for i in range(index, len(nums)):

if len(temp) > 0 and temp[-1] > nums[i]: continue

if nums[i] in used: continue

used[nums[i]] = True

temp.append(nums[i])

self.subsets(nums, i+1, temp, res)

temp.pop()

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

series =[4,6,7,7]

print("输入序列为：",series)

print("输出序列为：",s.findSubsequences(series))

【例228】神奇字符串

3.代码实现

class Solution(object):

def magicalString(self, n):

#参数n为整数

#返回整数

if n == 0:

return 0

elif n <= 3:

return 1

else:

so\_far, grp, ones =[1,2,2], 2, 1

while len(so\_far) < n:

freq, item = so\_far[grp], 1 if grp % 2 == 0 else 2

for \_ in range(freq):

so\_far.append(item)

ones, grp = ones + freq if item == 1 else ones, grp + 1

if len(so\_far) == n:

return ones

else:

return ones-1 if so\_far[-1] == 1 else ones

#主函数

if \_\_name\_\_ == '\_\_main\_\_':  
 s = Solution()  
 n = 6  
 print("输入为：",n)

print("输出为：",s.magicalString(n))

【例229】爆破气球的最小箭头数

3.代码实现

class Solution(object):

def findMinArrowShots(self, points):

#参数points为整数列表

#返回整数

if points == None or not points:

return 0

points.sort(key = lambda x : x[1]);

ans = 1

lastEnd = points[0][1]

for i in range(1, len(points)):

if points[i][0] > lastEnd:

ans += 1

lastEnd = points[i][1]

return ans

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n = [[10,16], [2,8], [1,6], [7,12]]

print("输入为：",n)

print("输出为：",s.findMinArrowShots(n))

【例230】查找数组中的所有重复项

3.代码实现

class Solution:

#参数nums为整数列表

#返回整数列表

def findDuplicates(self, nums):

if not nums:

return []

duplicates = []

for each in range(len(nums)):

index = nums[each]

if index<0:

index = -index

if nums[index-1]>0:

nums[index-1]=-nums[index-1]

else:

duplicates.append(index)

return duplicates

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n = [4,3,2,7,8,2,3,1]

print("输入为：",n)

print("输出为：",s.findDuplicates(n))

【例231】最小基因变化

3.代码实现

from collections import deque

class Solution:

#参数start为字符串

#参数end为字符串

#参数bank为字符串

#返回整数

def minMutation(self, start, end, bank):

if not bank:

return -1

bank = set(bank)

h = deque()

h.append((start, 0))

while h:

seq, step = h.popleft()

if seq == end:

return step

for c in "ACGT":

for i in range(len(seq)):

new\_seq = seq[:i] + c + seq[i + 1:]

if new\_seq in bank:

h.append((new\_seq, step + 1))

bank.remove(new\_seq)

return -1

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n ="AACCGGTT"

m= "AACCGGTA"

p=["AACCGGTA"]

print("输入起点为：",n)

print("输入终点为：",m)

print("输入的库为：",p)

print("输出步数为：",s.minMutation(n,m,p))

【例232】替换后的最长重复字符

3.代码实现

from collections import defaultdict

class Solution:

#参数s为字符串

#参数k为整数

#返回整数

def characterReplacement(self, s, k):

n = len(s)

char2count = defaultdict(int)

maxLen = 0

start = 0

for end in range(n):

char2count[s[end]] += 1

while end - start + 1 - char2count[s[start]] > k:

char2count[s[start]] -= 1

start += 1

maxLen = max(maxLen, end - start + 1)

return maxLen

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n ="ABAB"

m=2

print("输入字符串为：",n)

print("输入重复次数：",m)

print("输出子串长度：",s.characterReplacement(n,m))

【例233】从英文中重建数字

3.代码实现

class Solution:

#参数s为字符串

#返回字符串

def originalDigits(self, s):

nums = [0 for x in range(10)]

nums[0] = s.count('z')

nums[2] = s.count('w')

nums[4] = s.count('u')

nums[6] = s.count('x')

nums[8] = s.count('g')

nums[3] = s.count('h') - nums[8]

nums[7] = s.count('s') - nums[6]

nums[5] = s.count('v') - nums[7]

nums[1] = s.count('o') - nums[0] - nums[2] - nums[4]

nums[9] = (s.count('n') - nums[1] - nums[7]) // 2

result = ""

for x in range(10):

result += str(x) \* nums[x]

return result

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n ="owoztneoer"

print("输入为：",n)

print("输出为：",s.originalDigits(n))

【例234】数组中两个数字的最大异或

3.代码实现

class TrieNode:

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

self.val = val

self.children = {}

class Solution:

#参数nums为整数:

#返回整数

def findMaximumXOR(self, nums):

answer = 0

for i in range(32)[::-1]:

answer <<= 1

prefixes = {num >> i for num in nums}

answer += any(answer^1 ^ p in prefixes for p in prefixes)

return answer

def findMaximumXOR\_TLE(self, nums):

root = TrieNode(0)

for num in nums:

self.addNode(root, num)

res = -sys.maxsize

for num in nums:

cur\_node, cur\_sum = root, 0

for i in reversed(range(0,32)):

bit = (num >> i) & 1

if (bit^1) in cur\_node.children:

cur\_sum += 1 << i

cur\_node = cur\_node.children[bit^1]

else:

cur\_node = cur\_node.children[bit]

res = max(res, cur\_sum)

return res

def addNode(self, root, num):

cur = root

for i in reversed(range(0,32)):

bit = (num >> i) & 1

if bit not in cur.children:

cur.children[bit] = TrieNode(bit)

cur = cur.children[bit]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n =[3, 10, 5, 25, 2, 8]

print("输入为：",n)

print("输出为：",s.findMaximumXOR(n))

【例235】根据身高重排队列

3.代码实现

class Solution:

"""

参数people为整数列表

返回整数列表

"""

def reconstructQueue(self, people):

queue = []

for person in sorted(people, key = lambda \_: (-\_[0], \_[1])): queue.insert(person[1], person)

return queue

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n =[[7,0], [4,4], [7,1], [5,0], [6,1], [5,2]]

print("输入为：",n)

print("输出为：",s.reconstructQueue(n))

【例236】左叶子的和

3.代码实现

class TreeNode:  
 def \_\_init\_\_(self, val):  
 self.val = val  
 self.left, self.right = None, None

class Solution(object):

def sumOfLeftLeaves(self, root):

"""

参数root为二叉树根

返回整数

"""

def dfs(root):

if not root:

return 0

sum = 0

if root.left:

left = root.left;

#当前节点的左子节点,并判断是否为叶子节点

if not left.left and not left.right:

sum += left.val;

else :

sum += dfs(left)

if root.right:

right = root.right

sum += dfs(right)

return sum

return dfs(root)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

t = TreeNode(3)

t1 = TreeNode(9)

t.left = t1

t2 = TreeNode(20)

t.right = t2

t3 = TreeNode(15)

t2.left = t3

t4 = TreeNode(7)

t2.right = t4

print("输入为：[3,9 20,# # 15 7]")

print("输出为：",s.sumOfLeftLeaves(t))

【例237】移除K位

3.代码实现

class Solution:

"""

参数num为字符串

参数k为整数

返回字符串

"""

def removeKdigits(self, num, k):

if k == 0:

return num

if k >= len(num):

return "0"

result\_list = []

for i in range(len(num)):

while len(result\_list) > 0 and k > 0 and result\_list[-1] > num[i]:

result\_list.pop()

k -= 1

if num[i] != '0' or len(result\_list) > 0 :

result\_list.append(num[i])

while len(result\_list) > 0 and k > 0:

result\_list.pop()

k -= 1

if len(result\_list) == 0:

return '0'

return ''.join(result\_list)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n = "1432219"

k = 3

print("输入数字为：",n)

print("输入移除数：",k)

print("输出为：",s.removeKdigits(n,k))

【例238】轮转函数

3.代码实现

class Solution:

"""

参数A数组

返回整数

"""

def maxRotateFunction(self, A):

s = sum(A)

curr = sum(i\*a for i, a in enumerate(A))

maxVal = curr

for i in range(1, len(A)):

curr += s - len(A)\*A[-i]

maxVal = max(maxVal, curr)

return maxVal

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n = [4,3,2,6]

print("输入为：",n)

print("输出为：",s.maxRotateFunction(n))

【例239】字符至少出现K次的最长子串

3.代码实现

class Solution:

"""

参数s为字符串

参数k为整数

返回整数

"""

def longestSubstring(self, s, k):

for c in set(s):

if s.count(c) < k:

return max(self.longestSubstring(t, k) for t in s.split(c))

return len(s)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n = "aaabb"

k = 3

print("输入字符串为：",n)

print("输入重复次数：",k)

print("输出子串长度：",s.longestSubstring(n,k))

【例240】消除游戏

3.代码实现

class Solution:

"""

参数n为整数

返回整数

"""

def lastRemaining(self, n):

return 1 if n == 1 else 2 \* (1 + n // 2 - self.lastRemaining(n // 2))

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n = 9

print("输入为：",n)

print("输出为：",s.lastRemaining(n))

【例241】有序矩阵中的第K小元素

3.代码实现

class Solution:

"""

参数matrix为整数列表

参数k为整数

返回整数

"""

def kthSmallest(self, matrix, k):

start = matrix[0][0]

end = matrix[-1][-1]

while start + 1 < end:

mid = start + (end - start) // 2

if self.get\_num\_less\_equal(matrix, mid) < k:

start = mid

else:

end = mid

if self.get\_num\_less\_equal(matrix, start) >= k:

return start

return end

def get\_num\_less\_equal(self, matrix, mid):

m = len(matrix)

n = len(matrix[0])

i = 0

j = n - 1

count = 0

while i < m and j >= 0:

if matrix[i][j] <= mid:

i += 1

count += j + 1

else:

j -= 1

return count

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n=[[ 1, 5, 9],[10, 11, 13],[12, 13, 15]]

k=8

print("输入数组为：",n)

print("输入顺序为：",k)

print("输出数字为：",s.kthSmallest(n,k))

【例242】超级幂次

3.代码实现

class Solution:

"""

参数a为整数: the given number a

参数b为数组

返回整数

"""

def superPow(self, a, b):

if a == 0:

return 0

ans = 1

def mod(x):

return x % 1337

for num in b:

ans = mod(mod(ans \*\* 10) \* mod(a \*\* num))

return ans

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n=2

k=[3]

print("输入a=：",n)

print("输入b=：",k)

print("输出为：",s.superPow(n,k))

【例243】水罐问题

3.代码实现

class Solution:

"""

参数x为整数

参数y为整数

参数z为整数

返回布尔类型

"""

def canMeasureWater(self, x, y, z):

if x+y < z:

return False

return z % self.gcd(x,y) == 0

def gcd(self, x, y):

if y == 0:

return x

return self.gcd(y, x%y)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

x = 3

y = 5

z = 4

print("输入x=：",x)

print("输入y=：",y)

print("输入z=：",z)

print("输出为：",s.canMeasureWater(x,y,z))

【例244】计算不同数字整数的个数

3.代码实现

class Solution:

"""

参数n为整数

返回整数

"""

def countNumbersWithUniqueDigits(self, n):

if n == 0:

return 1

n = min(n, 10)

dp = [0] \* (n + 1)

dp[0], dp[1] = 1, 9

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

dp[i] = dp[i - 1] \* (11 - i)

return sum(dp)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

x = 2

print("输入为：",x)

print("输出为：",s.countNumbersWithUniqueDigits(2))

4.运行结果

输入为：2

输出为：91

【例245】最大乘积路径

3.代码实现

#参数x，y代表每条边的起始和终止

#参数d是每个节点的权重

#返回值是一个整数，是取模后节点最大乘积

class Solution:

ans = 0

def dfs(self, x, f, g, d, mul):

isLeaf = True

mul = mul \* d[x - 1] % 1000000007

for i in g[x]:

if i == f:

continue

isLeaf = False

self.dfs(i, x, g, d, mul)

if(isLeaf is True):

self.ans = max(self.ans, mul)

def getProduct(self, x, y, d):

g = [ [] for i in range(len(d) + 1)]

for i in range(len(x)):

g[x[i]].append(y[i])

g[y[i]].append(x[i])

self.dfs(1, -1, g, d, 1)

return self.ans

if \_\_name\_\_ == '\_\_main\_\_':

x =[1,2,2]

y = [2,3,4]

d = [1,1,-1,2]

solution = Solution()

print(" 每个边的起始和终止为：", x, y)

print(" 每个节点的权重为：", d)

print(" 最大路径上的乘积是:", solution.getProduct(x, y, d))

【例246】矩阵找数

3.代码实现

#参数mat是待查矩阵

#返回值是一个整数，是每一行都出现的最小的数字

class Solution:

def findingNumber(self, mat):

hashSet = {}

n = len(mat)

for mati in mat:

vis = {}

for x in mati:

vis[x] = 1

for key in vis:

if key not in hashSet:

hashSet[key] = 0

hashSet[key] += 1

ans = 100001

for i in hashSet:

if hashSet[i] == n:

ans = min(i, ans)

return -1 if ans == 100001 else ans

class Solution:

def findingNumber(self, mat):

hashSet = {}

n = len(mat)

for mati in mat:

vis = {}

for x in mati:

vis[x] = 1

for key in vis:

if key not in hashSet:

hashSet[key] = 0

hashSet[key] += 1

ans = 100001

for i in hashSet:

if hashSet[i] == n:

ans = min(i, ans)

return -1 if ans == 100001 else ans

if \_\_name\_\_ == '\_\_main\_\_':

mat = [[1,2,3],[3,4,1],[2,1,3]]

solution = Solution()

print(" 矩阵为：", mat)

print(" 每一行都出现的最小的数是：", solution.findingNumber(mat))

【例247】路径数计算

3.代码实现

#参数points是除了始末点外的必经点

#参数l和w是长和宽

#返回值是一个整数，有多少种走法

class Point:

def \_\_init\_\_(self, a=0, b=0):

self.x = a

self.y = b

class Solution:

def calculationTheSumOfPath(self, l, w, points):

points.sort(key = lambda point: point.x)

if points[0].x != 1 or points[0].y != 1:

points = [Point(1,1)] + points

if points[0].x != l or points[0].y != w:

points = points + [Point(l,w)]

arr = [[] for i in range(len(points) - 1)]

maxl = 0

maxw = 0

for i in range(1, len(points)):

l = points[i].x - points[i - 1].x

w = points[i].y - points[i - 1].y

arr[i - 1] = [points[i].x - points[i - 1].x, points[i].y - points[i - 1].y]

maxl = max(maxl, l)

maxw = max(maxw, w)

dp = [[0 for i in range(max(maxl, maxw) + 1)]for j in range(max(maxl, maxw) + 1)]

del l, w, maxl, maxw

for i in range(len(dp)):

for j in range(i, len(dp)):

if i == 0:

dp[j][i] = dp[i][j] = 1

else:

dp[j][i] = dp[i][j] = dp[i - 1][j] + dp[i][j - 1]

ans = 1

for i in arr:

ans = ans \* dp[i[0]][i[1]] % 1000000007

return ans

if \_\_name\_\_ == '\_\_main\_\_':

l=43

w=48

points = [Point(17,19), Point(43,48), Point(3,5)]

solution = Solution()

print(" 长与宽分别为：", l, w)

print(" 有路径种数:", solution.calculationTheSumOfPath(l, w, points))

【例248】卡牌游戏

3.代码实现

class Solution:

def numOfPlan(self, n, totalProfit, totalCost, a, b):

dp = [[0 for j in range(110)] for i in range(110)]

dp[0][0] = 1

mod = 1000000007

for i in range(n):

for j in range(totalProfit + 1, -1, -1):

for k in range(totalCost + 1, -1, -1):

idxA = min(totalProfit + 1, j + a[i])

idxB = min(totalCost + 1, k + b[i])

dp[idxA][idxB] = (dp[j][k] + dp[idxA][idxB]) % mod

ans = 0

for i in range(totalCost):

ans = (ans + dp[totalProfit + 1][i]) % mod

return ans

if \_\_name\_\_ == '\_\_main\_\_':

n = 2

totalProfit = 3

totalCost = 5

a = [2,3]

b = [2,2]

solution = Solution()

print(" 总卡片数：", n)

print(" 成本和利润的列表是：", a, b)

print(" 总成本是：", totalProfit, " 需要总利润是：", totalCost)

print(" 可使用方法总数:", solution.numOfPlan(n, totalProfit, totalCost, a, b))

【例249】词频统计

3.代码实现

#参数s是待查句子

#参数excludeList是被排除的单词

#返回值是一个字符串的列表，是所有出现频次最高的单词

class Solution:

def getWords(self, s, excludeList):

s = s.lower()

words = []

p = ''

for letter in s:

if letter < 'a' or letter > 'z':

if p != '':

words.append(p)

p = ''

else:

p += letter

if p != '':

words.append(p)

dic = {}

for word in words:

if word in dic:

dic[word] += 1

else:

dic[word] = 1

ans = []

mx = 0

for word in words:

if dic[word] > mx and (not word in excludeList):

mx = dic[word]

ans = [word]

elif dic[word] == mx and word not in ans and not word in excludeList:

ans.append(word)

return ans

if \_\_name\_\_ == '\_\_main\_\_':

s="Do do do do not not Trouble trouble."

excludeList=["do"]

solution = Solution()

print(" 待查句子为：",s , "除外词表为:", excludeList)

print(" 词频最高的单词为:", solution.getWords(s, excludeList))

【例250】查找子数组

3.代码实现

#参数arr是原数组

#参数k是目标子数组和

#返回值是个整数，代表这样一个子数组的起始位置，或者-1代表不存在

class Solution:

def searchSubarray(self, arr, k):

sum = 0

maps = {}

maps[sum] = 0

st = len(arr) + 5

lent = 0

for i in range(0, len(arr)):

sum += arr[i]

if sum - k in maps:

if st > maps[sum - k]:

st = maps[sum - k]

lent = i + 1 - maps[sum - k]

if sum not in maps:

maps[sum] = i + 1

if st == len(arr) + 5:

return -1

else:

return lent

if \_\_name\_\_ == '\_\_main\_\_':

arr = [1,2,3,4,5]

k = 5

solution = Solution()

print(" 数组为：", arr, "k为:", k)

print(" 最短和为k的子数组是:", solution.searchSubarray(arr, k))

【例251】最小子矩阵

3.代码实现

class Solution:

def minimumSubmatrix(self, arr):

ans = arr[0][0]

for i in range(len(arr)):

sum = [0 for x in range(len(arr[0]))]

for j in range(i,len(arr)):

for k in range(len(arr[0])):

sum[k] += arr[j][k]

dp = [0 for i in range(len(arr[0]))]

for k in range(len(arr[0])):

if k == 0:

dp[k] = sum[k]

else:

dp[k] = min(sum[k],dp[k-1]+sum[k])

ans = min(ans,dp[k])

return ans

if \_\_name\_\_ == '\_\_main\_\_':

arr = a = [[-3,-2,-1],[-2,3,-2],[-1,3,-1]]

solution = Solution()

print(" 数组为：", arr)

print(" 最小子数组是:", solution.minimumSubmatrix(arr))

【例252】最佳购物计划

3.代码实现

#参数k代表你有的钱

#参数m和n分别代表商品和礼盒数

#参数val是代表价值的列表

#参数costs是代表费用的列表

#返回值是一个整数，代表最大可获得价值

class Solution:

def getAns(self, n, m, k, val, cost, belong):

dp = [[-1 for i in range(0, 100001)] for i in range(0, 105)]

arr = [[] for i in range(0, 105)]

for i in range(n, n + m):

if not belong[i] == -1:

arr[belong[i]].append(i)

dp[0][cost[0]] = val[0]

for i in arr[0]:

for j in range(k, cost[i] - 1, -1):

if not dp[0][j - cost[i]] == -1:

dp[0][j] = dp[0][j - cost[i]] + val[i]

for i in range(1, n):

for j in range(k, cost[i] - 1, -1):

if not dp[i - 1][j - cost[i]] == -1:

dp[i][j] = dp[i - 1][j - cost[i]] + val[i]

dp[i][cost[i]] = val[i]

for j in arr[i]:

for l in range(k, cost[j] - 1, -1):

if not dp[i][l - cost[j]] == -1:

dp[i][l] = max(dp[i][l], dp[i][l - cost[j]] + val[j])

for j in range(0, k + 1):

dp[i][j] = max(dp[i][j], dp[i - 1][j])

ans = 0

for i in range(0, k + 1):

ans = max(ans, dp[n - 1][i])

return ans

if \_\_name\_\_ == '\_\_main\_\_':

k = 10

m = 2

n = 3

val = [17, 20, 8, 1, 4]

cost = [3, 5, 2, 3, 1]

belong = [-1, -1, -1, 0, 2]

solution = Solution()

print(" 拥有的钱：", k)

print(" 有商品数：", m, " 有礼盒数：", n)

print(" 价值的列表：", val, " 费用的列表：", cost)

print(" 可以得到最大价值:", solution.getAns(n, m, k, val, cost, belong))

【例253】询问冷却时间

3.代码实现

#参数arr是输入的待查数组

#参数n是公共冷却时间

#返回值是一个整数，代表需要多少时间

class Solution:

def askForCoolingTime(self, arr, n):

ans = 0

l = [0 for i in range(110)]

for x in arr:

if l[x] == 0 or ans - l[x] > n:

ans += 1

else:

ans = l[x] + n + 1

l[x] = ans

return ans

if \_\_name\_\_ == '\_\_main\_\_':

arr = [1, 2, 1, 2]

n = 2

solution = Solution()

print(" 数组为：", arr, " 冷却为：", n)

print(" 至少需要时间:", solution.askForCoolingTime(arr, n))

【例254】树上最长路径

3.代码实现

#参数n是节点总数

#参数starts是每条边的起始

#参数ends是每条边的结束

#参数lens是每条边的权重

#返回值是个整数，代表树上最长路径

import sys

sys.setrecursionlimit(200000)

class Solution:

G = []

dp = []

def dfs(self, u, pre):

for x in self.G[u]:

if x[0] != pre:

self.dp[x[0]] = self.dp[u] + x[1]

self.dfs(x[0], u)

def longestPath(self, n, starts, ends, lens):

self.G = [[] for i in range(n)]

self.dp = [0 for i in range(n)]

for i in range(n - 1):

self.G[starts[i]].append([ends[i], lens[i]])

self.G[ends[i]].append([starts[i], lens[i]])

self.dp[0] = 0

self.dfs(0, 0)

pos = Mx = 0

for i in range(n):

if self.dp[i] > Mx:

pos = i

Mx = self.dp[i]

self.dp[pos] = 0

self.dfs(pos, pos)

ans = 0

for i in range(n):

if self.dp[i] > ans:

ans = self.dp[i]

return ans

if \_\_name\_\_ == '\_\_main\_\_':

n = 5

starts = [0, 0, 2, 2]

ends = [1, 2, 3, 4]

lens = [1, 2, 5, 6]

solution = Solution()

print(" 总共有节点：", n)

print(" 每条边的起始为：", starts)

print(" 每条边的结束为：", ends)

print(" 每条边的权重为：", lens)

print(" 树上最长路径:", solution.longestPath(n, starts, ends, lens))

【例255】取数游戏

3.代码实现

#参数s和t是一对字符串，他们需要被验证能否互相根据规则转换

#返回值是个字符串，意为能否根据规则转换

class Solution:

def theGameOfTakeNumbers(self, arr):

n = len(arr)

if n == 0:

return 1

sum = [0 for i in range(n)]

for i in range(1, n + 1):

for j in range(0, n - i + 1):

if i == 1:

sum[j] = arr[j]

continue

k = j + i - 1

sum[j] = max(arr[k] - sum[j], arr[j] - sum[j + 1])

return 1 if sum[0] >= 0 else 2

if \_\_name\_\_ == '\_\_main\_\_':

arr = [1,3,3,1]

solution = Solution()

print(" 游戏数组为：", arr)

print(" 赢家会是:", solution.theGameOfTakeNumbers(arr))

【例256】数组求和

3.代码实现

#参数arr是原始总列表

#返回值是个整数，代表所有子数组的和

class Solution:

def findTheSumOfTheArray(self, arr):

ans = 0

n = len(arr)

for i in range(n):

ans = (ans + arr[i] \* (i + 1) \* (n - i)) % 1000000007;

return ans

if \_\_name\_\_ == '\_\_main\_\_':

arr = [2,4,6,8,10]

solution = Solution()

print(" 输入数组arr为：", arr)

print(" 所有子数组的和为:", solution.findTheSumOfTheArray(arr))

【例257】最短短语

3.代码实现

#参数k是最短单词数

#参数lim是最短短语长度

#参数str是被查找的字符串列表

#返回值是个整数，代表最短短语

class Solution:

def getLength(self, k, lim, str):

n = len(str)

arr = [0] \* n

for i in range(n):

arr[i] = len(str[i])

l = 0

r = 0

sum = 0

ans = 1e9

for r in range(n):

sum += arr[r]

while r - l >= k and sum - arr[l] >= lim:

sum -= arr[l]

l += 1

if r - l + 1 >= k and sum >= lim:

ans = min(ans, sum)

return ans

if \_\_name\_\_ == '\_\_main\_\_':

k = 2

lim = 10

str = ["she","was","bad","in","singing"]

solution = Solution()

print(" 最短单词数为：", k)

print(" 短语长度限制为大于：", lim)

print(" 文章列表为：", str)

print(" 最短短语为:", solution.getLength(k, lim, str))

【例258】频率最高的词

3.代码实现

#参数s为“小说”的字符串，excludeWords是不统计的词列表

#返回值是个单词，是出现最多的词

class Solution:

def frequentWord(self, s, excludewords):

map = {}

while len(s) > 0:

end = s.find(' ') if s.find(' ') > -1 else len(s)

word = s[:end] if s[end - 1].isalpha() else s[:end - 1]

s = s[end + 1:]

if word not in excludewords:

if word in map:

map[word] += 1

else:

map[word] = 1

max = -1

res = []

for key, val in map.items():

if val == max:

res.append(key)

elif val > max:

max = val

res = [key]

res.sort()

return res[0]

if \_\_name\_\_ == '\_\_main\_\_':

s = "Jimmy has an apple, it is on the table, he like it"

excludeWords = ["a","an","the"]

solution = Solution()

print("小说的内容为：", s)

print("统计不包含的词：", excludeWords)

print("最常出现的词为：", solution.frequentWord(s, excludeWords))

【例259】判断三角形

3.代码实现

#参数a为输入原始数组

#返回值是个字符串，意为能否形成三角形

class Solution:

def judgingTriangle(self, arr):

n = len(arr)

if n > 44:

return "YES"

arr.sort();

for i in range(n - 2):

for j in range(i + 1, n - 1):

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

if arr[i] + arr[j] > arr[k]:

return "YES"

return "NO"

if \_\_name\_\_ == '\_\_main\_\_':

a = [1,2,5,9,10]

solution = Solution()

print(" 输入数组为：", a)

print(" 能否组成三角形:", solution.judgingTriangle(a))

【例260】最大矩阵边界和

3.代码实现

#参数arr为输入矩阵

#返回值是个整数，代表最大边界数值的和

class Solution:

def solve(self, arr):

n = len(arr)

m = len(arr[0])

preCol = []

preRow = []

for r in range(n):

tem = [0]

res = 0

for c in range(m):

res += arr[r][c]

tem.append(res)

preRow.append(tem)

for c in range(m):

tem = [0]

res = 0

for r in range(n):

res += arr[r][c]

tem.append(res)

preCol.append(tem)

ans = arr[0][0]

for r1 in range(n):

for r2 in range(r1, n):

for c1 in range(m):

for c2 in range(c1, m):

if r1 == r2 and c1 == c2:

res = arr[r1][c1]

elif r1 == r2:

res = preRow[r1][c2 + 1] - preRow[r1][c1]

elif c1 == c2:

res = preCol[c1][r2 + 1] - preCol[c1][r1]

else:

res = preCol[c1][r2 + 1] - preCol[c1][r1] + preCol[c2][r2 + 1] - preCol[c2][r1] + \

preRow[r1][c2 + 1] - preRow[r1][c1] + preRow[r2][c2 + 1] - preRow[r2][c1] - arr[r1][

c1] - arr[r1][c2] - arr[r2][c1] - arr[r2][c2]

ans = max(ans, res)

return ans

if \_\_name\_\_ == '\_\_main\_\_':

arr=[[-1,-3,2],[2,3,4],[-3,7,2]]

solution = Solution()

print(" 矩阵为：", arr)

print(" 最大能得到边界和为：", solution.solve(arr))

【例261】卡牌游戏

3.代码实现

#参数Cost和Damage为卡牌属性

#参数totalCost和totalDamage为手上的费用和需要造成的伤害

#返回值是个布尔值，代表能否达成伤害

class Solution:

def cardGame(self, cost, damage, totalMoney, totalDamage):

num = len(cost)

dp = [0] \* (totalMoney + 1)

for i in range(0, num):

for j in range(totalMoney, cost[i] - 1, -1):

dp[j] = max(dp[j], dp[j - cost[i]] + damage[i])

if dp[j] >= totalDamage:

return True

return False

if \_\_name\_\_ == '\_\_main\_\_':

cost = [3,4,5,1,2]

damage = [3,5,5,2,4]

totalMoney = 7

totalDamage = 11

solution = Solution()

print(" 卡牌的cost和damage数组分别为：", cost, damage)

print(" 总共拥有金钱：", totalMoney)

print(" 需要造成伤害：", totalDamage)

print(" 能否达成：", solution.cardGame(cost, damage, totalMoney, totalDamage))

【例262】停车问题

3.代码实现

#参数a是进出表

#返回值是个整数，代表最多同时有几辆车

class Solution:

def getMax(self, a):

ans = [0] \* 23

for i in a:

for j in range(i[0], i[1]):

ans[j] += 1

max = ans[0]

for i in ans:

if i > max:

max = i

return max

if \_\_name\_\_ == '\_\_main\_\_':

a = [[1,2],[2,3],[3,4],[4,5]]

solution = Solution()

print(" 车辆进出表为：", a)

print(" 最多同时有车:", solution.getMax(a))

【例263】爬楼梯

3.代码实现

#参数a与b分别是匹配数组和价值数组

#返回值是个整数，代表选择区间的最大价值

class Solution:

def getAnswer(self, n, num):

ans = [0] \* (len(num) + 1)

ans[0] = 1

for i in range(n):

for j in range(1 + i, min(len(num) + 1, i + num[i] + 1)):

ans[j] = (ans[j] + ans[i]) % (10\*\*9 + 7)

return ans[len(num)]

if \_\_name\_\_ == '\_\_main\_\_':

n = 4

num = [1,1,1,1]

solution = Solution()

print(" 台阶数和每层台阶能往上登的阶数为：", n, num)

print(" 走到顶一共有几种走法：", solution.getAnswer(n, num))

【例264】最小字符串

3.代码实现

#参数s是原始字符串

#参数k是最大删除数目

#返回值是个字符串，代表删完的最小字典序字符串

class Solution:

def findMinC(self, s, k):

ans = 0

if len(s) <= k:

return -1

for i in range(1, k + 1):

if ord(s[i]) < ord(s[i - 1]):

ans = i

return ans

def MinimumString(self, s, k):

ans = ""

while k > 0:

temp = self.findMinC(s, k)

if temp == -1:

s = ''

break

ans = ans + s[temp]

s = s[temp + 1:]

k -= temp

ans += s

return ans

if \_\_name\_\_ == '\_\_main\_\_':

s = "cba"

k = 2

solution = Solution()

print(" 原始字符串为：", s)

print(" 可以删除到最小字典序:", solution.MinimumString(s, k))

【例265】目的地的最短路径

3.代码实现

#参数targetMap是地图

#返回值是个整数，是最短步数

class Solution:

ans = []

def cal(self, targetMap, x, y, z):

if targetMap[x][y] == 1:

return

if z < self.ans[x][y] or self.ans[x][y] == -1:

self.ans[x][y] = z

if x != 0:

self.cal(targetMap, x - 1, y, z + 1)

if x != len(targetMap) - 1:

self.cal(targetMap, x + 1, y, z + 1)

if y != 0:

self.cal(targetMap, x, y - 1, z + 1)

if y != len(targetMap[0]) - 1:

self.cal(targetMap, x, y + 1, z + 1)

return

def shortestPath(self, targetMap):

self.ans = [[-1 for i in range(len(targetMap[0]))] for j in range(len(targetMap))]

self.cal(targetMap, 0, 0, 0)

print(self.ans)

for i in range(len(targetMap)):

for j in range(len(targetMap[0])):

if targetMap[i][j] == 2:

return self.ans[i][j]

if \_\_name\_\_ == '\_\_main\_\_':

targetMap = [[0, 0, 0],[0, 0, 1],[0, 0, 2]]

solution = Solution()

print(" 地图为：", targetMap)

print(" 最少需要走:", solution.shortestPath(targetMap))

【例266】毒药测试

3.代码实现

#参数n是总水瓶数

#返回值是个整数，代表需要多少小白鼠

class Solution:

def getAns(self, n):

n -= 1

ans = 0

while n != 0:

n //= 2

ans += 1

return ans

if \_\_name\_\_ == '\_\_main\_\_':

n = 4

solution = Solution()

print("总共有：",n,"瓶水")

print("至少需要:", solution.getAns(n),"只小白鼠")

【例267】社交网络

3.代码实现

#参数n是网络人数

#参数a与b是关系两方

#返回值是个字符串，根据所有人能否联系返回“YES”或“NO”

class Solution:

father = [0] \* 5000

def ask(self, x):

if Solution.father[x] == x:

return x

Solution.father[x] = Solution.ask(self, Solution.father[x])

return Solution.father[x]

def socialNetwork(self, n, a, b):

for i in range(0, n):

Solution.father[i] = i

m = len(b)

for i in range(m):

x = Solution.ask(self, a[i])

y = Solution.ask(self, b[i])

Solution.father[x] = y

for i in range(0, n):

if Solution.ask(self, i) != Solution.ask(self, 1):

return "NO"

return "YES"

if \_\_name\_\_ == '\_\_main\_\_':

n = 4

a = [1, 1, 1]

b = [2, 3, 4]

solution = Solution()

print("好友关系组为：",a,b)

print("他们能否直接或间接互相联系：", solution.socialNetwork(n, a, b))

【例268】前K高的基点

3.代码实现

#参数list是学生ID与成绩的列表

#返回值是个列表，为前K名GPA学生的原序列表

from heapq import heappush, heappop

class Solution:

def topKgpa(self, list, k):

if len(list) == 0 or k < 0:

return []

minheap = []

ID\_set = set([])

result = []

for ID, GPA in list:

ID\_set.add(ID)

heappush(minheap, (float(GPA), ID))

if len(ID\_set) > k:

\_, old\_ID = heappop(minheap)

ID\_set.remove(old\_ID)

for ID, GPA in list:

if ID in ID\_set:

result.append([ID, GPA])

return result

if \_\_name\_\_ == '\_\_main\_\_':

List = [["001","4.53"],["002","4.87"],["003","4.99"]]

k = 2

solution = Solution()

print("学生按ID排序为：",List,"，K为：",k)

print("前K高GPA的学生为：", solution.topKgpa(List, k))

【例269】寻找最长01子串

3.代码实现

#参数str是原始01串

#返回值为整数，为最大长度

class Solution:

def askingForTheLongest01Substring(self, str):

str += str

ans = 1

cnt = 1

for i in range(1, len(str)):

if str[i] != str[i - 1]:

cnt += 1

else:

cnt = 1

if ans < cnt and 2 \* cnt <= len(str):

ans = cnt

return ans

if \_\_name\_\_ == '\_\_main\_\_':

str = "1001"

solution = Solution()

print(" 二进制串是",str)

print(" 最长01子串有:", solution.askingForTheLongest01Substring(str))

【例270】合法字符串

3.代码实现

#参数S是原始字符串

#参数k表示相同字符至少间隔多少字符

#返回值为列表，为每个位置插入的个数

class Solution:

def getAns(self, k, S):

n = len(S)

pre = [-1] \* 26 # 当前位置之前最靠右的相同字母位置,只有大写

sm = [0] \* (n + 1) #当前位置之前的"\_"总数

ans = []

for i in range(1, n + 1):

c = ord(S[i - 1]) - ord('A')

if pre[c] == -1 or sm[i - 1] - sm[pre[c]] - pre[c] + i >= k:

sm[i] = sm[i - 1]

ans.append(0)

else:

sm[i] = sm[i - 1] + k - (sm[i - 1] - sm[pre[c]] + i - pre[c])

ans.append(k - (sm[i - 1] - sm[pre[c]] + i - pre[c]))

pre[c] = i

return ans

if \_\_name\_\_ == '\_\_main\_\_':

S = "AABACCDCD"

k = 3

solution = Solution()

print(" 字符串是", S, "，每个相同字符间至少间隔",k , "个字符")

print(" \_字符的列表是:", solution.getAns(k,S))

【例271】叶节点的和

3.代码实现

#参数root是树根

#返回值为整数，为叶节点值的和

class TreeNode:

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

self.val = val

self.left, self.right = None, None

class Solution:#莫里斯中序遍历

def sumLeafNode(self, root):

res = 0

p = root

while p:

if p.left is None:

if p.right is None: # p 是一个叶节点

res += p.val

p = p.right

else:

tmp = p.left

while tmp.right is not None and tmp.right != p:

tmp = tmp.right

if tmp.right is None:

if tmp.left is None: # tmp 是一个叶子节点

res += tmp.val

tmp.right = p

p = p.left

else: # 因为tmp.right为前序，所以停止

tmp.right = None

p = p.right

return res

if \_\_name\_\_ == '\_\_main\_\_':

n1 = TreeNode(1)

n1.left = TreeNode(2)

n1.right = TreeNode(3)

n1.left.left = TreeNode(4)

n1.left.right = TreeNode(5)

solution = Solution()

print(" 结果为:", solution.sumLeafNode(n1))

【例272】转换字符串

3.代码实现

#参数startString是起始链

#参数endString是目标链

#返回值是布尔值，如果可以转换则返回True，否则返回False

class Solution:

def canTransfer(self, startString, endString):

if not startString and not endString:

return True

# 长度不等

if len(startString) != len(endString):

return False

# 字母种类起始链比终止链少

if len(set(startString)) < len(set(endString)):

return False

maptable = {}

for i in range(len(startString)):

a, b = startString[i], endString[i]

if a in maptable:

if maptable[a] != b:

return False

else:

maptable[a] = b

def noloopinhash(maptable): # 映射表带环

keyset = set(maptable)

while keyset:

a = keyset.pop()

loopset = {a}

while a in maptable:

if a in keyset:

keyset.remove(a)

loopset.add(a)

if a == maptable[a]:

break

a = maptable[a]

if a in loopset:

return False

return True

return noloopinhash(maptable)

if \_\_name\_\_ == '\_\_main\_\_':

startString = "abc"

endString = "bca"

solution = Solution()

print(" 起始链是：", startString)

print(" 终止链是：", endString)

print(" 能否转换：", solution.canTransfer(startString, endString))

【例273】最少按键次数

3.代码实现

#参数s是一个字符串

#返回值为整数，为最小按键次数

class Solution:

def getAns(self, s):

left = -1

ans = 0

ncaps = True

for right in range(0, len(s)):

if ncaps:

if ord(s[right]) < 95 and right-left <= 2:

ans += 2

if right-left == 2:

ncaps = False

ans -= 1

left = right

else:

left = right

ans +=1

else:

if ord(s[right]) > 95 and right-left <= 2:

ans += 2

if right - left == 2:

ncaps = True

ans -= 1

left = right

else:

left = right

ans +=1

return ans

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

str = "EWlweWXZXxcscSDSDcccsdcfdsFvccDCcDCcdDcGvTvEEdddEEddEdEdAs"

solution = Solution()

print(" str是：", str)

print(" 最小按键数是:", solution.getAns(str))

【例274】二分查找

3.代码实现

class Solution:

# 参数nums为整数数组

# 参数target为整数

# 返回整数

def binarySearch(self, nums, target):

left, right = 0, len(nums)-1

while left + 1 < right :

mid = (left + right)// 2

if nums[mid] < target :

left = mid

else :

right = mid

if nums[left] == target :

return left

elif nums[right] == target :

return right

return -1;

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

nums=[1,3,4,5,6,9]

target=6

solution = Solution()

answer=solution.binarySearch(nums,target)

print("输入数组为：",nums)

print("输入目标为：",target)

print("输出下标为：",answer)

【例275】全排列

3.代码实现

class Solution:

"""

参数nums为整数列表

返回排序列表

"""

def permute(self, nums):

if nums is None:

return []

if nums == []:

return [[]]

nums = sorted(nums)

permutation = []

stack = [-1]

permutations = []

while len(stack):

index = stack.pop()

index += 1

while index < len(nums):

if nums[index] not in permutation:

break

index += 1

else:

if len(permutation):

permutation.pop()

continue

stack.append(index)

stack.append(-1)

permutation.append(nums[index])

if len(permutation) == len(nums):

permutations.append(list(permutation))

return permutations

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

nums=[0,1,2]

name=solution.permute(nums)

print("输入为：",nums)

print("输出为：",name)

【例276】最小路径和

3.代码实现

class Solution:

"""

参数grid为整数列表

返回整数

"""

def minPathSum(self, grid):

for i in range(len(grid)):

for j in range(len(grid[0])):

if i == 0 and j > 0:

grid[i][j] += grid[i][j-1]

elif j == 0 and i > 0:

grid[i][j] += grid[i-1][j]

elif i > 0 and j > 0:

grid[i][j] += min(grid[i-1][j], grid[i][j-1])

return grid[len(grid) - 1][len(grid[0]) - 1]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

nums=[[1,3,1],[1,5,1],[4,2,1]]

answer=solution.minPathSum(nums)

print("输入列表为：",nums)

print("输出路径和：",answer)

【例277】最长路径序列

3.代码实现

class Solution:

"""

参数num为整数列表

返回整数

"""

def longestConsecutive(self, num):

dict = {}

for x in num:

dict[x] = 1

ans = 0

for x in num:

if x in dict:

len = 1

del dict[x]

l = x - 1

r = x + 1

while l in dict:

del dict[l]

l -= 1

len += 1

while r in dict:

del dict[r]

r += 1

len += 1

if ans < len:

ans = len

return ans

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

nums=[100,4,200,1,3,2]

answer=solution.longestConsecutive(nums)

print("输入列表为：",nums)

print("输出长度为：",answer)

【例278】背包问题2

3.代码实现

class Solution:

# 参数m为整数

# 参数A和V为整数列表

def backPackII(self, m, A, V):

n = len(A)

dp = [[0] \* (m + 1), [0] \* (m + 1)]

for i in range(1, n + 1):

dp[i % 2][0] = 0

for j in range(1, m + 1):

dp[i % 2][j] = dp[(i - 1) % 2][j]

if A[i - 1] <= j:

dp[i % 2][j] = max(dp[i % 2][j], dp[(i - 1) % 2][j - A[i - 1]] + V[i - 1])

return dp[n % 2][m]

# 主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

vol=34

nums=[4,13,2,6,7,11,8]

val=[1,23,4,5,2,14,9]

answer = solution.backPackII(vol,nums,val)

print("输入总体积：",vol)

print("输入物品为：",nums)

print("输入价值为：",val)

print("输出结果为：",answer)

【例279】哈希函数

3.代码实现

class Solution:

"""

参数key为字符串

参数HASH\_SIZE为整数

返回整数

"""

def hashCode(self, key, HASH\_SIZE):

ans = 0

for x in key:

ans = (ans \* 33 + ord(x)) % HASH\_SIZE

return ans

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

num=100

key="abcd"

answer= solution.hashCode(key,num)

print("输入key为：",key)

print("输入num为：",num)

print("输出值为：",answer)

【例280】第一个只出现一次的字符

3.代码实现

class Solution:

"""

参数str为字符串

返回字符

"""

def firstUniqChar(self, str):

counter = {}

for c in str:

counter[c] = counter.get(c, 0) + 1

for c in str:

if counter[c] == 1:

return c

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

s = "abaccdeff"

ans = solution.firstUniqChar(s)

solution = Solution()

s = "abaccdeff"

ans = solution.firstUniqChar(s)

print("输入为：", s)

print("输出为：", ans)

【例281】空格替换

3.代码实现

class Solution:

# 参数string为字符数组

# 参数length为字符串的真实长度

# 返回新字符串的真实长度

def replaceBlank(self, string, length):

if string is None:

return length

f = 0

L = len(string)

for i in range(len(string)):

if string[i] == ' ':

string[i] = '%20'

f+=1

return L-f+f\*3

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

si = "Mr John Smith"

s1 = list(si)

ans = solution.replaceBlank(s1, 13)

so = ''.join(s1)

print("输入字符串：", si)

print("输出字符串：", so)

print("输出其长度：", ans)

【例282】字符串压缩

3.代码实现

class Solution:

"""

参数originalString为字符串

返回压缩字符串

"""

def compress(self, originalString):

l = len(originalString)

if l <=2 :

return originalString

length = 1

res = ""

for i in range(1,l):

if originalString[i] != originalString[i-1]:

res = res + originalString[i-1] + str(length)

length = 1

else:

length += 1

if originalString[-1] != originalString[-2]:

res = res + originalString[-1] + "1"

else:

res = res + originalString[i-1] + str(length)

if len(originalString)<=len(res):

return originalString

else:

return res

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

si = "aabcccccaaa"

arr = list(si)

ans = solution.compress(arr)

print("输入为：", si)

print("输出为：", ans)

【例283】数组的最大值

3.代码实现

class Solution:

def max\_num(self, arr):

if arr == []:

return

maxnum = arr[0]

for x in arr:

if x > maxnum:

maxnum = x

return maxnum

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

arr = [1.0, 2.1, -3.3]

ans = solution.max\_num(arr)

print("输入为：", arr)

print("输出为：", ans)

【例284】无序链表的重复项删除

3.代码实现

class ListNode(object):

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

self.val = val

self.next = None

class Solution:

"""

    参数head为链表的第一个节点。

    返回头结点

"""

def removeDuplicates(self, head):

seen, root, pre = set(), head, ListNode(-1)

while head:

if head.val not in seen:

pre.next = head

seen.add(head.val)

pre = head

head = head.next

pre.next = None

return root

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

l0 = ListNode(1)

l1 = ListNode(2)

l2 = ListNode(2)

l3 = ListNode(2)

l0.next = l1

l1.next = l2

l2.next = l3

root = solution.removeDuplicates(l0)

a = [root.val, root.next.val]

if a == [1, 2]:

print("输入： 1->2->2->2->null")

print("输出： 1->2->null")

else:

print("Error")

【例285】在O(1)时间复杂度删除链表节点

3.代码实现

#参数node是要删除的节点

#无返回值，操作完毕

class ListNode(object):

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

self.val = val

self.next = next

class Solution:

def deleteNode(self, node):

if node.next is None:

node = None

return

node.val = node.next.val

node.next = node.next.next

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

node1=ListNode(1)

node2=ListNode(2)

node3=ListNode(3)

node4=ListNode(4)

node1.next=ListNode(2)

node2.next=ListNode(3)

node3.next=ListNode(4)

solution= Solution()

print("输入是 :",node1.val,node2.val,node3.val,node4.val)

solution.deleteNode(node3)

print("删除节点3")

print("输出是 :",node1.val,node2.val,node3.val)

【例286】将数组重新排序以构造最小值

3.代码实现

from functools import cmp\_to\_key

class Solution:

def cmp(self,a,b):

if a+b>b+a:

return 1

if a+b<b+a:

return -1

else:

return 0

def PrintMinNumber(self,numbers):

if not numbers:

return ""

number = list(map(str,numbers))

number.sort(key=cmp\_to\_key(self.cmp))

return "".join(number).lstrip('0') or '0'

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generation=[3,32,321]

solution= Solution()

print("输入是 :",generation)

print("输出是 :",solution.PrintMinNumber(generation))

【例287】两个链表的交叉

3.代码实现

#参数list\_a是一个链表

#参数list\_b是另一个链表

#无返回值，直接打印出结果

class ListNode:

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

self.value = val

self.next = next

class Solution:

def get\_list\_length(self, head):

"""获取链表长度"""

length = 0

while head:

length += 1

head = head.next

return length

def get\_intersect\_node(self, list\_a, list\_b):

length\_a = self.get\_list\_length(list\_a)

length\_b = self.get\_list\_length(list\_b)

cur1, cur2 = list\_a, list\_b

if length\_a > length\_b:

for i in range(length\_a - length\_b):

cur1 = cur1.next

else:

for i in range(length\_b - length\_a):

cur2 = cur2.next

flag = False

while cur1 and cur2:

if cur1.value == cur2.value:

print(cur1.value)

flag = True

break

else:

cur1 = cur1.next

cur2 = cur2.next

if not flag:

print('链表没有交叉结点')

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution= Solution()

list\_a = ListNode('a1', ListNode('a2', ListNode('c1', ListNode('c2', ListNode('c3')))))

list\_b = ListNode('b1', ListNode('b2', ListNode('b3', ListNode('c1', ListNode('c2', ListNode('c3'))))))

print("输入是：")

print("a = a1 a2 c1 c2 c3")

print("b = b1 b2 b3 c1 c2 c3")

print("输出是：")

solution.get\_intersect\_node(list\_a,list\_b)

【例288】螺旋矩阵

3.代码实现

#参数n是指123 … n任意一个整型数

#返回值是一个矩阵

class Solution:

def generateMatrix(self, n):

if n == 0: return []

matrix = [[0 for i in range(n)] for j in range(n)]

up = 0; down = len(matrix)-1

left = 0; right = len(matrix[0])-1

direct = 0; count = 0

while True:

if direct == 0:

for i in range(left, right+1):

count += 1; matrix[up][i] = count

up += 1

if direct == 1:

for i in range(up, down+1):

count += 1; matrix[i][right] = count

right -= 1

if direct == 2:

for i in range(right, left-1, -1):

count += 1; matrix[down][i] = count

down -= 1

if direct == 3:

for i in range(down, up-1, -1):

count += 1; matrix[i][left] = count

left += 1

if count == n\*n: return matrix

direct = (direct+1) % 4

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

n=3

solution = Solution()

print("输入是： n = ", n)

print("输出是：",solution.generateMatrix(n))

【例289】三角形计数

3.代码实现

#参数S是一个正整数数组

#返回值count是计数结果

class Solution:

def triangleCount(self, S):

if len(S)<3:

return;

count=0;

S.sort();#从小到大排序

for i in range(0,len(S)):

for j in range(i+1,len(S)):

w,r=i+1,j

target=S[j]-S[i]

while w<r:

mid=(w +r)//2 #取整数

S\_mid=S[mid]

if S\_mid>target:

r=mid

else:

w=mid+1

count+=(j-w)

return count

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generation=[3,4,6,7]

solution= Solution()

print("输入是：", generation)

print("输出是：",solution.triangleCount(generation))

【例290】买卖股票的最佳时机

3.代码实现

class Solution:

"""

参数k为整数

参数prices为整数数组

返回整数

"""

def maxProfit(self, k, prices):

size = len(prices)

if k >= size / 2:

return self.quickSolve(size, prices)

dp = [-10000] \* (2 \* k + 1)

dp[0] = 0

for i in range(size):

for j in range(min(2 \* k, i + 1) , 0 , -1):

dp[j] = max(dp[j], dp[j - 1] + prices[i] \* [1, -1][j % 2])

return max(dp)

def quickSolve(self, size, prices):

sum = 0

for x in range(size - 1):

if prices[x + 1] > prices[x]:

sum += prices[x + 1] - prices[x]

return sum

#主函数

if \_\_name\_\_ == "\_\_main\_\_":

solution=Solution()

price=[4,4,6,1,1,4,2,5]

k=2

maxprofit=solution.maxProfit(k,price)

print("输入价格为：",price)

print("交易次数为：",k)

print("最大利润为：",maxprofit)

【例291】加一

3.代码实现

class Solution:

#参数digits为整数数组

#返回整数数组

def plusOne(self, digits):

digits = list(reversed(digits))

digits[0] += 1

i, carry = 0, 0

while i < len(digits):

next\_carry = (digits[i] + carry) // 10

digits[i] = (digits[i] + carry) % 10

i, carry = i + 1, next\_carry

if carry > 0:

digits.append(carry)

return list(reversed(digits))

#主函数

if \_\_name\_\_ =="\_\_main\_\_":

solution=Solution()

num=[9,9,9]

answer=solution.plusOne(num)

print("输入为：",num)

print("输出为：",answer)

【例292】炸弹袭击

3.代码实现

#参数grid是一个表示二维网格的数组，由'W' 'E' '0'组成

#返回值result是放置一个炸弹后消灭敌人的最大数量

class Solution:

def maxKilledEnemies(self, grid):

m, n = len(grid), 0

if m:

n = len(grid[0])

result, rows = 0, 0

cols = [0 for i in range(n)]

for i in range(m):

for j in range(n):

if j == 0 or grid[i][j-1] == 'W':

rows = 0

for k in range(j, n):

if grid[i][k] == 'W':

break

if grid[i][k] == 'E':

rows += 1

if i == 0 or grid[i-1][j] == 'W':

cols[j] = 0

for k in range(i, m):

if grid[k][j] == 'W':

break

if grid[k][j] == 'E':

cols[j] += 1

if grid[i][j] == '0' and rows + cols[j] > result:

result = rows + cols[j]

return result

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generation =[

"0E00",

"E0WE",

"0E00"

]

solution= Solution()

print("输入是：", generation)

print("输出是：", solution.maxKilledEnemies(generation))

【例293】组合总和 IV

3.代码实现

#参数nums是一个不重复的正整型数组

#参数target是一个整数

#返回值是一个整数，表示组合方式的个数

class Solution:

def backPackVI(self, nums, target):

row = len(nums)

col = target

dp = [0 for i in range(col + 1)]

dp[0] = 1

for j in range(1, col + 1):

for i in range(1, row + 1):

if nums[i - 1] > j:

continue

dp[j] += dp[j - nums[i - 1]]

return dp[-1]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generation=[1,2,4]

target=4

solution= Solution()

print("输入是：", generation)

print("输出是：", solution.backPackVI(generation,target))

【例294】向循环有序链表插入节点

3.代码实现

#参数node是要插入的链表节点序列

#参数x是一个整数，表示插入的新的节点

#返回值new\_node是插入新节点后的链表序列

class ListNode:

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

self.val = val

self.next = next

class Solution:

def insert(self, node, x):

new\_node = ListNode(x)

if node is None:

node = new\_node

node.next = node

return node

#定义当前节点和前一节点

cur, pre = node, None

while cur:

pre = cur

cur = cur.next

# pre.val <= x <= cur.val

if x <= cur.val and x >= pre.val:

break

#链表循环处特殊判断（最大值->最小值），如果x小于最小值或x大于最大值，在此插入

if pre.val > cur.val and (x < cur.val or x > pre.val):

break

#循环了一遍

if cur is node:

break

#插入该节点

new\_node.next = cur

pre.next = new\_node

return new\_node

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

k=4

generation = ListNode(3, ListNode(5, ListNode(1)))

solution= Solution()

solution.insert(generation,k)

print("输入是： {3,5,1}")

print("输出是：",generation.val,generation.next.val,generation.next.next.val, generation.next.next.next.val)

【例295】大岛的数量

3.代码实现

class Solution:

"""

参数grid为二维布尔数组

参数k为整数

返回岛的数量

"""

def numsofIsland(self, grid, k):

# Write your code here

if not grid or len(grid)==0 or len(grid[0])==0: return 0

rows, cols = len(grid), len(grid[0])

visited = [[False for i in range(cols)] for i in range(rows)]

res = 0

for i in range(rows):

for j in range(cols):

if visited[i][j]==False and grid[i][j] == 1:

check = self.bfs(grid, visited, i,j,k)

if check: res+=1

return res

def bfs(self, grid, visited, x, y, k):

rows, cols = len(grid), len(grid[0])

import collections

queue = collections.deque([(x, y)])

visited[x][y] = True

res = 0

while queue:

item = queue.popleft()

res+=1

for idx, idy in ((1,0),(-1,0),(0,1),(0,-1)):

x\_new, y\_new = item[0]+idx, item[1]+idy

if x\_new < 0 or x\_new >= rows or y\_new < 0 or y\_new >= cols or visited[x\_new][y\_new] or grid[x\_new][y\_new] == 0: continue

queue.append((x\_new, y\_new))

visited[x\_new][y\_new] = True

return res >= k

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

g = [[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]]

k = 3

ans = solution.numsofIsland(g, k)

print("输入：", g, "\nk=", k)

print("输出：", ans)

【例296】最短回文串

3.代码实现

class Solution:

"""

参数str为字符串

返回字符串

"""

def convertPalindrome(self, str):

if not str or len(str) == 0:

return ""

n = len(str)

for i in range(n - 1, -1, -1):

substr = str[:i + 1]

if self.isPalindrome(substr):

if i == n - 1:

return str

else:

return (str[i + 1:] [::-1]) + str[:]

def isPalindrome(self, str):

left, right = 0, len(str) - 1

while left < right:

if str[left] != str[right]:

return False

left += 1

right -= 1

return True

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

s = "sdsdlkjsaoio"

ans = solution.convertPalindrome(s)

print("输入：", s)

print("输出：", ans)

【例297】不同的路径

3.代码实现

class Solution:

"""

参数grid为二维数组

返回所有唯一加权路径之和

"""

def uniqueWeightedPaths(self, grid):

n=len(grid)

m=len(grid[0])

if n == 0 or m == 0:

return 0

s=[[set() for \_ in range(m)] for \_\_ in range(n)]

s[0][0].add(grid[0][0])

for i in range(n):

for j in range(m):

if i==0 and j==0:

s[i][j].add(grid[i][j])

else:

for val in s[i-1][j]:

s[i][j].add(val+grid[i][j])

for val in s[i][j-1]:

s[i][j].add(val+grid[i][j])

ans=0

for val in s[-1][-1]:

ans+=val

return ans

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

arr = [[1,1,2],[1,2,3],[3,2,4]]

ans = solution.uniqueWeightedPaths(arr)

print("输入：", arr)

print("输出：", ans)

【例298】分割字符串

3.代码实现

class Solution:

"""

参数s为要拆分的字符串

返回所有可能的拆分字符串数组

"""

def splitString(self, s):

result = []

self.dfs(result, [], s)

return result

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

if s == "":

result.append(path[:]) #important: use path[:] to clone it

return

for i in range(2):

if i+1 <= len(s):

path.append(s[:i+1])

self.dfs(result, path, s[i+1:])

path.pop()

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

s = "123"

ans = solution.splitString(s)

print("输入：", s)

print("输出：", ans)

【例299】缺失的第一个素数

3.代码实现

class Solution:

"""

参数nums为数组

返回整数

"""

def firstMissingPrime(self, nums):

if not nums:

return 2

start = 0

l = len(nums)

integer = 2

while start < l:

while self.isPrime(integer) == False:

integer += 1

if nums[start] != integer:

return integer

integer += 1

start += 1

while self.isPrime(integer) == False:

integer += 1

return integer

def isPrime(self, num):

if num == 2 or num == 3:

return True

for i in range(2, int(num\*\*(0.5)) + 1):

if num % i == 0:

return False

return True

if \_\_name\_\_=='\_\_main\_\_':

solution=Solution()

n=[3,5,7]

print("输入为：",n)

print("输出为：",solution.firstMissingPrime(n))

【例300】单词拆分

3.代码实现

class Solution:

"""

参数s为字符串

参数dict为单词列表

返回整数数量

"""

def wordBreak3(self, s, dict):

if not s or not dict:

return 0

n, hash = len(s), set()

lowerS = s.lower()

for d in dict:

hash.add(d.lower())

f = [[0] \* n for \_ in range(n)]

for i in range(n):

for j in range(i, n):

sub = lowerS[i:j + 1]

if sub in hash:

f[i][j] = 1

for i in range(n):

for j in range(i, n):

for k in range(i, j):

f[i][j] += f[i][k] \* f[k + 1][j]

return f[0][-1]

if \_\_name\_\_=='\_\_main\_\_':

solution=Solution()

s="CatMat"

dict1=["Cat", "Mat", "Ca", "tM", "at", "C", "Dog", "og", "Do"]

print("输入句子为：",s)

print("输入列表为：",dict1)

print("输出数量为：",solution.wordBreak3(s,dict1))

例301 单例模式

3.代码实现

class Solution:

#返回这个类的同一个实例

instance = None

@classmethod

def getInstance(cls):

if cls.instance is None:

cls.instance = Solution()

return cls.instance

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

a = temp.getInstance()

b = temp.getInstance()

print('输入: "a=temp.getInstance()" 和 "b=temp.getInstance()"')

print('输出: a和b是否得到同一个实例？' + str(a == b))

例302 字符串置换

3.代码实现

class Solution:

#参数str1和str2是需要判断的两个字符串

#返回布尔型，两个字符串是否可以置换

def permutation(self, str1, str2):

s1 = "".join((lambda x: (x.sort(), x)[1])(list(str1)))

s2 = "".join((lambda x: (x.sort(), x)[1])(list(str2)))

if s1 == s2:

return True

else:

return False

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

str1 = 'abcdd'

str2 = 'dbcad'

str3 = 'acd'

str4 = 'abc'

print('输入: str1=' + str1 + ' str2=' + str2)

print('输出: ' + str(temp.permutation(str1, str2)))

print('输入: str3=' + str3 + ' str4=' + str4)

print('输出: ' + str(temp.permutation(str3, str4)))

例303 字符串替换

3.代码实现

class Solution:

#参数string为字符数组

#返回压缩后的字符串

def compressString(self, string):

l = len(string)

if l <= 2:

return string

length = 1

res = ''

for i in range(1, l):

if string[i] != string[i - 1]:

res = res + string[i - 1] + str(length)

length = 1

else:

length += 1

if string[-1] != string[-2]:

res = res + string[-1] + '1'

else:

res = res + string[-1] + str(length)

if len(string) <= len(res):

return string

else:

return res

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

str1 = 'aaabbccccdde'

str2 = 'hellooo'

print('输入: '+str1)

print('输出: '+temp.compressString(str1))

print('输入: ' + str2)

print('输出: ' + temp.compressString(str2))

例304 用isSubstring判断字符串的循环移动

3.代码实现

class Solution:

#s1为第一个字符串

#s2 为第二个字符串

#返回布尔型

def isRotation(self, s1, s2):

if len(s1) != len(s2) or len(s1) == 0:

return False

s1s1 = s1 + s1

return self.isSubstring(s1s1, s2)

def isSubstring(self, s, t):

return s.find(t) != -1

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

s1 = 'abcdef'

s2 = 'defabc'

s3 = 'abcefg'

s4 = 'efgacb'

print('输入: s1=' + s1 + ' s2=' + s2)

print('输出: ' + str(temp.isRotation(s1, s2)))

print('输入: s3=' + s3 + ' s4=' + s4)

print('输出: ' + str(temp.isRotation(s3, s4)))

例305 能否到达终点

3.代码实现

import queue as Queue

DIRECTIONS = [(-1, 0), (1, 0), (0, 1), (0, -1)]

SAPCE = 1

OBSTACLE = 0

ENDPOINT = 9

class Solution:

#参数map是一个地图

#返回一个布尔值，判断是否能到达终点

def reachEndpoint(self, map):

if not map or not map[0]:

return False

self.n = len(map)

self.m = len(map[0])

queue = Queue.Queue()

queue.put((0, 0))

while not queue.empty():

curr = queue.get()

for i in range(4):

x = curr[0] + DIRECTIONS[i][0]

y = curr[1] + DIRECTIONS[i][1]

if not self.isValid(x, y, map):

continue

if map[x][y] == ENDPOINT:

return True

queue.put((x, y))

map[x][y] = OBSTACLE

return False

def isValid(self, x, y, map):

if x < 0 or x >= self.n or y < 0 or y >= self.m:

return False

if map[x][y] == OBSTACLE:

return False

return True

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

map = [[1, 1, 1], [1, 1, 1], [1, 1, 9]]

print("地图：", map)

solution = Solution()

print("能否到达终点", solution.reachEndpoint(map))

例306 成绩等级

3.代码实现

class Student:

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

self.name = name

self.score = 0

def getLevel(self):

if self.score >= 90:

return 'A'

elif self.score >= 80:

return 'B'

elif self.score >= 60:

return 'C'

else:

return 'D'

if \_\_name\_\_ == '\_\_main\_\_':

student1 = Student('Jack')

student1.score = 60

student2 = Student('Ama')

student2.score = 95

print('输入: 学生1 Jack,成绩60')

print('输出: 学生等级' + student1.getLevel())

print('输入: 学生2 Ama,成绩95')

print('输出: 学生等级' + student2.getLevel())

例307 在排序链表中插入一个节点

3.代码实现

class ListNode(object):

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

self.val = val

self.next = next

class Solution:

#参数head为链表节点

#参数val为要插入的整数

#返回值为新链表的节点

def insertNode(self, head, val):

dummy = ListNode(0, head)

p = dummy

while p.next and p.next.val < val:

p = p.next

node = ListNode(val, p.next)

p.next = node

return dummy.next

def getLinkedList(head):

list = []

while head is not None:

list += [str(head.val)]

head = head.next

list.append('null')

s = '->'.join(list)

return s

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

LinkedNode1 = ListNode(1, ListNode(2, ListNode(3, ListNode(3, ListNode(5, ListNode(7))))))

val1 = 4

LinkedNode2 = ListNode(1, ListNode(3, ListNode(9, ListNode(11, ListNode(12, ListNode(15))))))

val2 = 13

print('输入: ' + getLinkedList(LinkedNode1) + ' val1=' + str(val1))

print('输出: ' + getLinkedList(temp.insertNode(LinkedNode1, val1)))

print('输入: ' + getLinkedList(LinkedNode2) + ' val2=' + str(val2))

print('输出: ' + getLinkedList(temp.insertNode(LinkedNode2, val2)))

例308 Getter与Setter

3.代码实现

class School:

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

self.\_\_name = ''

def setName(self, name):

self.\_\_name = name

def getName(self):

return self.\_\_name

if \_\_name\_\_ == '\_\_main\_\_':

school1 = School()

school1.setName('MIT')

school2 = School()

school2.setName('UIUC')

print('输入: school1 name= ' + school1.getName())

print('输出: school2 name= ' + school2.getName())

例309 用一个数组实现三个栈

3.代码实现

class ThreeStacks:

#参数size为整型, 每个栈的大小

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

self.size = size

self.stacks = [[], [], []]

#stack\_num为整数，第几个栈

#value为整型，放入栈的数值

def push(self, stack\_num, value):

self.stacks[stack\_num].append(value)

#stack\_num为整型，第几个栈

#返回整数，栈中存在的值

def pop(self, stack\_num):

return self.stacks[stack\_num].pop()

#参数stack\_num为整数，第几个栈

#返回整数，栈中存在的值

def peek(self, stack\_num):

return self.stacks[stack\_num][-1]

#参数stack\_num为整数，第几个栈

#返回布尔类型，栈是否为空

def isEmpty(self, stack\_num):

return len(self.stacks[stack\_num]) == 0

if \_\_name\_\_ == '\_\_main\_\_':

temp = ThreeStacks(5)

temp.push(0, 5)

temp.push(0, 10)

temp.push(1, 5)

a = temp.pop(0)

b = temp.pop(1)

temp.push(2, 10)

temp.push(2, 3)

c = temp.pop(2)

d = temp.peek(2)

e = temp.isEmpty(1)

f = temp.isEmpty(0)

print('''输入:temp=ThreeStacks(5)

temp.push(0,5)

temp.push(0,10)

temp.push(1,5)

temp.pop(0)

temp.pop(1)

temp.push(2,10)

temp.push(2,3)

temp.pop(2)

temp.peek(2)

temp.isEmpty(1)

temp.isEmpty(0)''')

print('输出:temp.pop(0) =' + str(a) + ' temp.pop(1) =' + str(b))

print(' temp.pop(2) =' + str(c) + ' temp.peek(2) =' + str(d))

print(' temp.isEmpty(1) =' + str(e) + ' temp.isEmpty(0) =' + str(f))

例310 在链表中找节点

3.代码实现

class ListNode(object):

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

self.val = val

self.next = next

class Solution:

#参数head为链表的头节点

#参数val为需要查找的数值

#返回值为一个节点或None

def findNode(self, head, val):

i = 0

while head is not None:

i = i + 1

if head.val == val:

return i

head = head.next

return None

def getLinkedList(head):

list = []

while head is not None:

list += [str(head.val)]

head = head.next

list.append('null')

s = '->'.join(list)

return s

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

head1 = ListNode(1, ListNode(2, ListNode(3, ListNode(5))))

val1 = 2

head2 = ListNode(2, ListNode(4, ListNode(6, ListNode(10))))

val2 = 3

print('输入: ' + getLinkedList(head1) + ' val1=' + str(val1))

print('输出: 第' + str(temp.findNode(head1, val1)) + '个节点')

print('输入: ' + getLinkedList(head2) + ' val1=' + str(val2))

print('输出: 第' + str(temp.findNode(head2, val2)) + '个节点')

例311 栈集

3.代码实现

class setOfStacks:

#capacity为整数，子栈的容量

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

self.stacks = []

self.capacity = capacity

#v为整数值

def push(self, v):

if len(self.stacks) == 0:

self.stacks.append([])

if len(self.stacks[-1]) == self.capacity:

self.stacks.append([])

self.stacks[-1].append(v)

#返回整数

def pop(self):

v = self.stacks[-1].pop()

if len(self.stacks[-1]) is 0:

self.stacks.pop()

return v

#返回子栈顶层元素的值

def popAt(self, index):

return self.leftShift(index, True)

def leftShift(self, index, removeTop):

if removeTop:

removed\_item = self.stacks[index][-1]

self.stacks[index].pop()

else:

removed\_item = self.stacks[index][0]

self.stacks[index].pop(0)

if len(self.stacks[index]) is 0:

self.stacks.pop(index)

elif len(self.stacks) > index + 1:

v = self.leftShift(index + 1, False)

self.stacks[index].append(v)

return removed\_item

if \_\_name\_\_ == '\_\_main\_\_':

temp = setOfStacks(2)

temp.push(1)

temp.push(2)

temp.push(4)

temp.push(8)

temp.push(16)

print('输入: temp.push(1) temp.push(2) temp.push(4) temp.push(8) temp.push(16)')

print('输出: [' + str(temp.popAt(0)) + ',' + str(temp.popAt(0)) + ',' + str(temp.pop()) + ']')

例312 链表的中点

3.代码实现

class ListNode(object):

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

self.val = val

self.next = next

class Solution:

#参数head为链表的头节点

#返回链表的中间节点

def middleNode(self, head):

if head is None:

return None

slow = head

fast = slow.next

while fast is not None and fast.next is not None:

slow = slow.next

fast = fast.next.next

return slow

def getLinkedList(head):

list = []

while head is not None:

list += [str(head.val)]

head = head.next

s = '->'.join(list)

return s

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

head1 = ListNode(1, ListNode(3, ListNode(5, ListNode(7, ListNode(9)))))

head2 = ListNode(1, ListNode(2, ListNode(3, ListNode(4))))

print('输入链表: ' + getLinkedList(head1))

print('输出中间值: ' + str(temp.middleNode(head1).val))

print('输入链表: ' + getLinkedList(head2))

print('输出中间值: ' + str(temp.middleNode(head2).val))

例313 栈排序

3.代码实现

class Solution:

#参数stack为栈的列表

#返回排序后栈的列表

def stackSort(self, stack):

temp = []

while len(stack):

if len(stack) and (not len(temp) or temp[-1] >= stack[-1]):

temp.append(stack.pop())

else:

value = stack.pop()

while len(temp) and temp[-1] <= value:

stack.append(temp.pop())

stack.append(value)

while len(temp):

stack.append(temp.pop())

while len(temp):

stack.append(temp.pop())

return stack

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

stack1 = [1, 2, 5, 9, 2, 3]

stack2 = [3, 4, 1, 3, 8, 7]

print('输入: ' + str(stack1))

print('输出：' + str(temp.stackSort(stack1)))

print('输入: ' + str(stack2))

print('输出：' + str(temp.stackSort(stack2)))

例314 宠物收养所

3.代码实现

class AnimalShelter(object):

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

self.cats = []

self.dogs = []

self.tot = 0

#参数name为字符串

#参数type为整型, 狗为1，猫为0

#无返回值

def enqueue(self, name, type):

self.tot += 1

if type == 1:

self.dogs.append([name, self.tot])

else:

self.cats.append([name, self.tot])

#返回资历最老的猫或狗

def dequeueAny(self):

if len(self.dogs) == 0:

return self.dequeueCats()

elif len(self.cats) == 0:

return self.dequeueDogs()

else:

if self.dogs[0][1] < self.cats[0][1]:

return self.dequeueDogs()

else:

return self.dequeueCats()

#返回要取出的狗名字

def dequeueDogs(self):

name = self.dogs[0][0]

del self.dogs[0]

return name

#返回要取出的猫名字

def dequeueCats(self):

name = self.cats[0][0]

del self.cats[0]

return name

if \_\_name\_\_ == '\_\_main\_\_':

temp = AnimalShelter()

temp.enqueue('Max', 1)

temp.enqueue('Mike', 0)

temp.enqueue('Ama', 1)

temp.enqueue('Anna', 0)

print("输入宠物信息: temp.enqueue('Max',1) temp.enqueue('Mike',0) temp.enqueue('Ama',1) temp.enqueue('Anna',0)")

print("领养宠物顺序: temp.dequeueCats() temp.dequeueDogs() temp.dequeueAny()")

print('输出宠物信息: [' + temp.dequeueCats() + ',' + temp.dequeueDogs() + ',' + temp.dequeueAny() + ']')

例315 自动补全

3.代码实现

class TypeAhead:

#参数dict为字符串列表

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

self.mp = {}

for s in dict:

l = len(s)

for i in range(l):

for j in range(i + 1, l + 1):

tmp = s[i:j]

if tmp not in self.mp:

self.mp[tmp] = [s]

elif self.mp[tmp][-1] != s:

self.mp[tmp].append(s)

#参数word为要查询的字符串

#返回值为自动补全的字符串

def search(self, word):

if word not in self.mp:

return []

else:

return self.mp[word]

if \_\_name\_\_ == '\_\_main\_\_':

dict = ["San Zhang", "Lisi", "Li Ma", "Jimmy Wang"]

temp = TypeAhead(dict)

print('输入: 查询 "Li"')

print('字典: '+str(dict[0:4]))

print('输出: '+str(temp.search('Li')))

例316 短网址

3.代码实现

class TinyUrl:

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

self.dict = {}

def getShortKey(self, url):

return url[-6:]

def idToShortKey(self, id):

ch = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"

s = ""

while id > 0:

s = ch[int(id % 62)] + s

id /= 62

while len(s) < 6:

s = 'a' + s

return s

def shortkeyToid(self, short\_key):

id = 0

for c in short\_key:

if 'a' <= c and c <= 'z':

id = id \* 62 + ord(c) - ord('a')

if 'A' <= c and c <= 'Z':

id = id \* 62 + ord(c) - ord('A') + 26

if '0' <= c and c <= '9':

id = id \* 62 + ord(c) - ord('0') + 52

return id

#参数url为字符串

#返回开头为http://tiny.url/的短url字符串

def longToShort(self, url):

ans = 0

for a in url:

ans = (ans \* 256 + ord(a)) % 56800235584

while ans in self.dict and self.dict[ans] != url:

ans = (ans + 1) % 56800235584

self.dict[ans] = url

return "http://tiny.url/" + self.idToShortKey(ans)

#参数url开头为 http://tiny.url/的短url

#返回值为一个长url

def shortToLong(self, url):

short\_key = self.getShortKey(url)

return self.dict[self.shortkeyToid(short\_key)]

if \_\_name\_\_ == '\_\_main\_\_':

temp = TinyUrl()

print('输入: shortToLong(longToShort("http://www.bupt.edu.cn"))')

print('输出: ' + temp.shortToLong(temp.longToShort("http://www.bupt.edu.cn")))

例317 拥有同样多1的下一个数

3.代码实现

class Solution:

#参数n为一个32位整数

#返回一个32位整数或-1

def getPrev(self, n):

temp = n;

c0 = 0;

c1 = 0;

while (temp & 1) == 1:

c1 += 1

temp >>= 1

if temp == 0:

return -1

while ((temp & 1) == 0) and (temp != 0):

c0 += 1

temp >>= 1

return n - (1 << c1) - (1 << (c0 - 1)) + 1

#参数n为一个32位整数

#返回值为一个32位整数或-1

def getNext(self, n):

temp = n

c0 = 0

c1 = 0

while ((temp & 1) == 0) and (temp != 0):

c0 += 1

temp >>= 1

while (temp & 1) == 1:

c1 += 1

temp >>= 1

result = n + (1 << c0) + (1 << (c1 - 1)) - 1;

if result < 0 or result >= (1 << 31):

return -1

return n + (1 << c0) + (1 << (c1 - 1)) - 1

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

print('输入: n1=5,n2=9')

print('输出: n1: Smaller:' + str(temp.getPrev(5)) + ',Larger:' + str(temp.getNext(5)))

print('输出: n2: Smaller:' + str(temp.getPrev(9)) + ',Larger:' + str(temp.getNext(9)))

例318 分解质因数

3.代码实现

import math

class Solution:

#参数num为整型

#返回值为整数列表

def primeFactorization(self, num):

up = int(math.sqrt(num)) + 1

f = [0 for x in range(up)]

prime = []

for i in range(2, up):

if f[i] == 0:

prime.append(i)

for j in range(i \* i, up, i):

f[j] = 1

rt = []

for a in prime:

while num % a == 0:

rt.append(a)

num /= a

if num != 1:

rt.append(int(num))

return rt

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

print('输入: 15')

print('输出: ')

print(temp.primeFactorization(15))

print('输入: 1250')

print('输出: ')

print(temp.primeFactorization(1250))

例319 交换奇偶二进制位

3.代码实现

class Soltuion:

#参数x为32位整数

#返回值为一个32位整数

def swapOddEvenBits(self, x):

return (((x & 0xaaaaaaaa) >> 1) | ((x & 0x55555555) << 1))

if \_\_name\_\_ == '\_\_main\_\_':

temp = Soltuion()

print('输入: n1=5, n2=10')

print('输出: ' + str(temp.swapOddEvenBits(5)) + ' ' + str(temp.swapOddEvenBits(10)))

例320 丢失的数

3.代码实现

class Solution:

#参数data为整数数组

#返回值为丢失的整数

def findMissingNumber(self, data):

n = len(data)

if not data:

return

result1 = 0

for value in data:

result1 ^= value

result2 = 0

for value in range(1, n + 1):

result2 ^= value

result = result1 ^ result2

return result

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

data1 = [1, 2, 0, 3, 5]

data2 = [0, 1, 3, 4, 5, 2, 6, 9, 8]

print('输入: ')

print(data1)

print(data2)

print('输出: ')

print(temp.findMissingNumber(data1))

print(temp.findMissingNumber(data2))

例321 黑白屏

3.代码实现

class Solution:

#参数screen为一个整数数组

#参数width为屏幕宽度，整型

#参数x1为起始位置，整型

#参数x2为终点位置，整型

#参数y为行数，整型

#返回整数数组

def drawHorizontalLine(self, screen, width, x1, x2, y):

start\_offset = x1 % 8

first\_full\_byte = x1 // 8

if start\_offset != 0:

first\_full\_byte += 1

end\_offset = x2 % 8

last\_full\_byte = x2 // 8

if end\_offset != 7:

last\_full\_byte -= 1

for b in range(first\_full\_byte, last\_full\_byte + 1):

screen[(width // 8) \* y + b] = 0xFF

start\_mask = 0xFF >> start\_offset

end\_mask = (~(0xFF >> (end\_offset + 1))) & 0xFF

if (x1 // 8) == (x2 // 8):

mask = start\_mask & end\_offset

screen[(width // 8) \* y + (x1 // 8)] |= mask

else:

if start\_offset != 0:

byte\_number = (width // 8) \* y + first\_full\_byte - 1

screen[byte\_number] |= start\_mask

if end\_offset != 7:

byte\_number = (width // 8) \* y + last\_full\_byte + 1

screen[byte\_number] |= end\_mask

return screen

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

screen = [0, 0, 0, 0]

width = 16

x1 = 4

x2 = 11

y = 1

print('输入初始屏幕:')

print(screen)

print('宽度: ' + str(width))

print('x1: ' + str(x1))

print('x2: ' + str(x2))

print('y: ' + str(y))

print('输出: ')

print(temp.drawHorizontalLine(screen, width, x1, x2, y))

例322 方程的根

3.代码实现

import math

class Solution:

#参数为a浮点型, 方程系数a

#参数为b浮点型, 方程系数b

#参数为c浮点型, 方程系数c

#返回值为浮点型的方程根

def rootOfEquation(self, a, b, c):

if b \* b - 4 \* a \* c < 0:

return []

if b \* b - 4 \* a \* c == 0:

return [-b \* 1.0 / 2 / a]

if b \* b - 4 \* a \* c > 0:

delta = math.sqrt(b \* b - 4 \* a \* c)

return sorted([(-b - delta) / (2 \* a), (-b + delta) / (2 \* a)])

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

a1 = 2

b1 = 16

c1 = 2

a2 = 4

b2 = 3

c2 = 3

print('输入:')

print('a1=' + str(a1) + ',b1=' + str(b1) + ',c1=' + str(c1))

print('输出:')

print(temp.rootOfEquation(a1, b1, c1))

print('输入:')

print('a2=' + str(a2) + ',b2=' + str(b2) + ',c2=' + str(c2))

print('输出:')

print(temp.rootOfEquation(a2, b2, c2))

例323 转换字符串到整数

3.代码实现

class Solution:

#参数为str 字符串

#返回值为整数

def stringToInteger(self, s):

num, sig = 0, 1

if s[0] == '-':

sig = -1

s = s[1:]

for c in s:

num = num \* 10 + ord(c) - ord('0')

return num \* sig

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

s1 = '1234'

s2 = '1357'

print('输入: s1="' + s1 + '"')

print('输出: ' + str(temp.stringToInteger(s1)))

print('输入: s2="' + s2 + '"')

print('输出: ' + str(temp.stringToInteger(s2)))

例324 将二叉树按照层级转化为链表

3.代码实现

class TreeNode:

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

self.val = val

self.left, self.right = left, right

if self.val == '#':

self.left = None

self.right = None

class ListNode:

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

self.val = val

self.next = next

class Solution:

#参数root为二叉树的根

#返回链表

def binaryTreeToLists(self, root):

result = []

if root is None:

return result

import queue

queue = queue.Queue()

queue.put(root)

dummy = ListNode(0)

lastNode = None

while not queue.empty():

dummy.next = None

lastNode = dummy

size = queue.qsize()

for i in range(size):

head = queue.get()

lastNode.next = ListNode(head.val)

lastNode = lastNode.next

if head.left is not None:

queue.put(head.left)

if head.right is not None:

queue.put(head.right)

result.append(dummy.next)

return result

def getLinkedList(result):

l = len(result)

ans = []

for i in range(l):

list = []

while result[i] is not None:

list += [str(result[i].val)]

result[i] = result[i].next

list.append('null')

s = '->'.join(list)

ans.append(s)

return ans

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

rootNode = TreeNode(1, TreeNode(3, TreeNode(2), TreeNode(5)), TreeNode(4, TreeNode(6), TreeNode(7)))

print('''输入:

1

/ \

3 4

/ \ / \

2 5 6 7 ''')

print('输出: ')

print(getLinkedList(temp.binaryTreeToLists(rootNode)))

例325 相亲数

3.代码实现

import math

class Solution:

#参数k为整数

#返回所有相亲数

def amicablePair(self, k):

result = []

for i in range(k + 1):

if self.d(self.d(i)) == i and self.d(i) < i:

result.append([self.d(i), i])

return result

def d(self, x):

sum = 1

p = int(math.sqrt(x))

for i in range(2, p):

if x % i == 0:

sum += i + x // i

if p \* p == x and p != 1:

sum += p

return sum

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

k = 300

print('输入: ')

print(k)

print('输出: ')

print(temp.amicablePair(k))

例326 二叉树的路径和I

3.代码实现

class TreeNode:

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

self.val = val

self.left, self.right = None, None

class Solution:

#root为二叉树的根节点

#target为期望路径和

#返回所有可能路径

def binaryTreePathSum(self, root, target):

result = []

path = []

self.dfs(root, path, result, 0, target)

return result

def dfs(self, root, path, result, len, target):

if root is None:

return

path.append(root.val)

len += root.val

if root.left is None and root.right is None and len == target:

result.append(path[:])

self.dfs(root.left, path, result, len, target)

self.dfs(root.right, path, result, len, target)

path.pop()

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

node1 = TreeNode(1)

node2 = TreeNode(2)

node3 = TreeNode(2)

node4 = TreeNode(3)

node5 = TreeNode(4)

node1.left = node2

node1.right = node5

node2.left = node3

node2.right = node4

target = 5

print('输入: {1,2,4,2,3} target=5')

print('输出: ')

print(temp.binaryTreePathSum(node1, target))

例327 二叉树的路径和 II

3.代码实现

class TreeNode:

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

self.val = val

self.left = left

self.right = right

class Solution:

#参数root为二叉树的根节点

#参数target为目标数

#返回所有可能路线

def binaryTreePathSum2(self, root, target):

path = []

result = []

if root is None:

return result

self.dfs(root, path, result, 0, target)

return result

def dfs(self, root, path, result, l, target):

if root is None:

return

path.append(root.val)

tmp = target

for i in range(l, -1, -1):

tmp -= path[i]

if tmp == 0:

result.append(path[i:])

self.dfs(root.left, path, result, l + 1, target)

self.dfs(root.right, path, result, l + 1, target)

path.pop()

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

rootNode = TreeNode(1, TreeNode(2, TreeNode(4)), TreeNode(3, TreeNode(2)))

target = 6

print('输入: {1,2,4,3,2} 目标值: 6')

print('输出: ')

print(temp.binaryTreePathSum2(rootNode, target))

例328 丢鸡蛋

3.代码实现

class Solution:

#参数n为楼层高

#返回最坏情况下需要的次数

def dropEggs(self, n):

import math

x = math.ceil((math.sqrt(8 \* n + 1) - 1) / 2)

return x

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

print('输入: 10')

print('输出: ' + str(temp.dropEggs(10)))

print('输入: 100')

print('输出: ' + str(temp.dropEggs(100)))

例329建立邮局

3.代码实现

#参数grid是一个二维的网格

#返回值是一个整数

from collections import deque

import sys

class Solution:

def shortestDistance(self, grid):

if not grid:

return 0

m = len(grid)

n = len(grid[0])

dist = [[sys.maxsize for j in range(n)] for i in range(m)]

reachable\_count = [[0 for j in range(n)] for i in range(m)]

min\_dist = sys.maxsize

buildings = 0

for i in range(m):

for j in range(n):

if grid[i][j] == 1:

self.bfs(grid, i, j, dist, m, n, reachable\_count)

buildings += 1

for i in range(m):

for j in range(n):

if reachable\_count[i][j] == buildings and dist[i][j] < min\_dist:

min\_dist = dist[i][j]

return min\_dist if min\_dist != sys.maxsize else -1

def bfs(self, grid, i, j, dist, m, n, reachable\_count):

visited = [[False for y in range(n)] for x in range(m)]

visited[i][j] = True

q = deque([(i, j, 0)])

while q:

i, j, l = q.popleft()

if dist[i][j] == sys.maxsize:

dist[i][j] = 0

dist[i][j] += l

for x, y in ((1, 0), (-1, 0), (0, 1), (0, -1)):

nx, ny = i + x, j + y

if -1 < nx < m and -1 < ny < n and not visited[nx][ny]:

visited[nx][ny] = True

if grid[nx][ny] == 0:

q.append((nx, ny, l + 1))

reachable\_count[nx][ny] += 1

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

grid = [[0, 1, 0, 0, 0], [1, 0, 0, 2, 1], [0, 1, 0, 0, 0]]

print("网格：", grid)

solution = Solution()

print("最近的距离：", solution.shortestDistance(grid))

例330 凑 N 分钱的方案数

3.代码实现

class Solution:

#参数n为要凑N分钱

#返回凑N分钱的不同方式数量

def waysNCents(self, n):

cents = [1, 5, 10, 25]

ways = [0 for \_ in range(n + 1)]

ways[0] = 1

for cent in cents:

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

ways[j] += ways[j - cent]

return ways[n]

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

n1 = 11

n2 = 6

print('输入: 要凑出' + str(n1) + '分钱')

print('输出: 有' + str(temp.waysNCents(n1)) + '种不同的方式')

print('输入: 要凑出' + str(n2) + '分钱')

print('输出: 有' + str(temp.waysNCents(n2)) + '种不同的方式')

例331 三数之中的最大值

3.代码实现

class Solution:

#参数a为整数

#参数b为整数

#参数c为整数

#返回三个数中的最大值

def maxOfThreeNumbers(self, a, b, c):

return max(a, b, c)

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

a = 4

b = 2

c = 3

print('输入: a=' + str(a) + ' b=' + str(b) + ' c=' + str(c))

print('输出: ' + str(temp.maxOfThreeNumbers(a, b, c)))

例332 接雨水

2.问题示例

给定一个 5\*4的矩阵：

[[12,13,0,12],[13,4,13,12],[13,8,10,12],[12,13,12,12],[13,13,13,13]]，如图3-1所示。

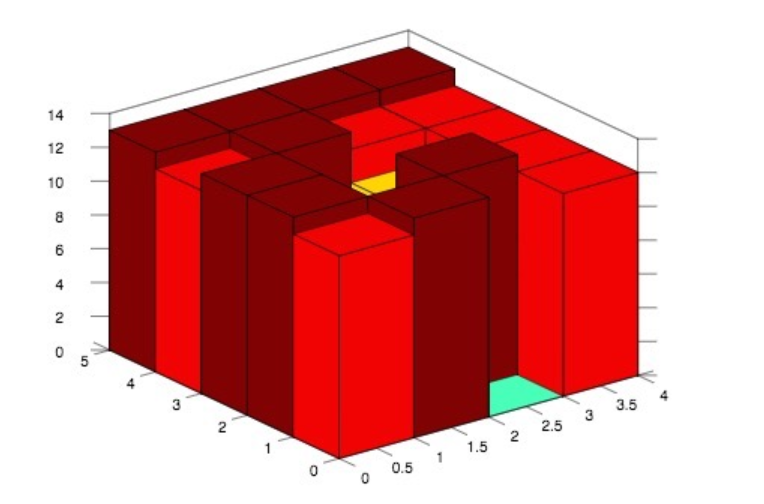


图3-1 接雨水示意图

矩阵周边的格子上面是无法盛水的，除周边格子，每次挑出一个高度最小的格子，与周围的格子相减，计算上面的盛水量。盛水量等于周围的格子高度减去这个格子的高度，如果这个值是负数，盛水量等于0。给定一个4\*4矩阵：

输入:[[2,2,2,2],[2,2,3,4],[3,3,3,1],[2,3,4,5]]

输出:0

例333 将表达式转换为波兰表达式

3.代码实现

class Solution:

#参数expression为表达式

#返回值为波兰表达式

def convertToPN(self, expression):

stk = []

PN = []

for s in expression[::-1]:

if s == ')':

stk.append(s)

elif s == '(':

pos = stk[::-1].index(')')

PN += stk[::-1][:pos]

stk = stk[:-pos - 1]

elif s[0] in '1234567890':

PN.append(s)

else:

priority = self.getPriority(s)

while len(stk) and self.getPriority(stk[-1]) > priority:

PN.append(stk[-1])

stk.pop()

stk.append(s)

PN += stk[::-1]

return PN[::-1]

def getPriority(self, s):

if s in '\*/':

return 3

if s in '+-':

return 2

if s in '()':

return 1

return 0

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

expression = ['(', '4', '-', '5', ')', '\*', '9']

print('输入: ')

print(expression)

print('输出: ')

print(temp.convertToPN(expression))

例334 将二叉树转换成双链表

3.代码实现

class DoublyListNode(object):

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

self.val = val

self.next = self.prev = next

class TreeNode:

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

self.val = val

self.left, self.right = None, None

class Solution():

#参数root为二叉树的根节点

#返回双向链表的头节点

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

self.cur = None

self.dummy = DoublyListNode(0)

def bstToDoublyList(self, root):

if root is None:

return None

self.cur = self.dummy

self.dfs(root)

self.dummy.next.prev = None

return self.dummy.next

def dfs(self, root):

if root is None:

return

self.dfs(root.left)

self.cur.next = DoublyListNode(root.val)

self.cur.next.prev = self.cur

self.cur = self.cur.next

self.dfs(root.right)

def getLinkedList(head):

list = []

while head is not None:

list += [str(head.val)]

head = head.next

list.append('null')

s = '<->'.join(list)

return s

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

node1 = TreeNode(1)

node2 = TreeNode(2)

node3 = TreeNode(3)

node4 = TreeNode(4)

node5 = TreeNode(5)

node6 = TreeNode(6)

node7 = TreeNode(7)

node1.left = node2

node1.right = node3

node2.left = node4

node2.right = node5

node3.left = node6

node3.right = node7

head = temp.bstToDoublyList(node1)

print('输入: {1,2,3,4,5,6,7}')

print('输出: ')

print(getLinkedList(head))

例335 将数组重新排列以构建最小值

3.代码实现

import functools

class Solution:

#参数nums为一个非负整数组

#返回排列后数组

def minNumber(self, nums):

nums.sort(key=functools.cmp\_to\_key(self.cmp))

return nums

def cmp(self, a, b):

if str(a) + str(b) < str(b) + str(a):

return -1

elif str(a) + str(b) == str(b) + str(a):

return 1

else:

return 0

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

nums = [3, 32, 321]

print('输入: ')

print(nums)

print('输出: ')

print(temp.minNumber(nums))

例336 动态数组ArrayList

3.代码实现

class arrayListManager:

#参数n为数组包含的元素个数

#返回创建的数组

def create(self, n):

list1 = []

for i in range(n):

list1.append(i)

return list1

#参数list为需要克隆的数组

#返回一个克隆数组

def clone(self, list):

dist = []

for a in list:

dist.append(a)

return dist

#参数list为被获取元素的数组

#参数k为需要获取的元素

#返回获取的元素

def get(self, list, k):

return list[k]

#参数list为数组

#参数k为数组中的元素

#参数val为修改的值

def set(self, list, k, val):

list[k] = val

#参数list为被删除元素的数组

#k为要删除的元素

def remove(self, list, k):

list.remove(k)

#参数list为数组

#参数val为需要被获取标识的元素

#返回元素的标识

def indexOf(self, list, val):

if list is None:

return -1

try:

ans = list.index(val)

except ValueError:

ans = -1

return ans

if \_\_name\_\_ == '\_\_main\_\_':

temp = arrayListManager()

array1 = temp.create(5)

val1 = temp.get([0, 1, 2, 3, 4], 0)

val2 = temp.get([0, 1, 2, 3, 4], 1)

val3 = temp.get([0, 1, 2, 3, 4], 4)

array2 = temp.clone([0, 1, 2, 3, 4])

val4 = temp.get([0, 1, 2, 3, 4], 2)

index1 = temp.indexOf([0, 1, 2, 3, 4], 1)

index2 = temp.indexOf([0, 1, 2, 3, 4], 10)

array3 = [0, 1, 2, 3, 4]

temp.remove(array3, 3)

val5 = temp.get([0, 1, 2, 4], 3)

array4 = [0, 1, 2, 4]

temp.set(array4, 2, 3)

val6 = temp.get([0, 1, 2, 3, 4], 2)

val7 = temp.get([0, 1, 2, 3, 4], 3)

print('输入: ')

print('''

create(5)

get([0,1,2,3,4], 0)

get([0,1,2,3,4], 1)

get([0,1,2,3,4], 4)

clone([0,1,2,3,4])

get([0,1,2,3,4], 2)

indexOf([0,1,2,3,4], 1)

indexOf([0,1,2,3,4], 10)

remove([0,1,2,3,4], 3)

get([0,1,2,4], 3)

set([0,1,2,4], 2, 3)

get([0,1,2,3,4], 2)

get([0,1,2,3,4], 3)

''')

print('输出:')

print(array1)

print(val1)

print(val2)

print(val3)

print(array2)

print(val4)

print(index1)

print(index2)

print(array3)

print(val5)

print(array4)

print(val6)

print(val7)

例337 找峰值

3.代码实现

import sys

class Solution:

#参数A为一个整数矩阵

#返回峰值的位置

def findPeakII(self, A):

if not A or not A[0]:

return None

return self.find\_peak(A, 0, len(A) - 1, 0, len(A[0]) - 1)

def find\_peak(self, matrix, top, bottom, left, right):

if top + 1 >= bottom and left + 1 >= right:

for row in range(top, bottom + 1):

for col in range(left, right + 1):

if self.is\_peak(matrix, row, col):

return [row, col]

return [-1, -1]

if bottom - top < right - left:

col = (right + left) // 2

row = self.find\_col\_peak(matrix, col, top, bottom)

if self.is\_peak(matrix, row, col):

return [row, col]

if matrix[row][col - 1] > matrix[row][col]:

return self.find\_peak(matrix, top, bottom, left, col - 1)

return self.find\_peak(matrix, top, bottom, left, col + 1)

row = (bottom + top) // 2

col = self.find\_row\_peak(matrix, row, left, right)

if self.is\_peak(matrix, row, col):

return [row, col]

if matrix[row - 1][col] > matrix[row][col]:

return self.find\_peak(matrix, top, row - 1, left, right)

return self.find\_peak(matrix, row + 1, bottom, left, right)

def is\_peak(self, matrix, x, y):

return matrix[x][y] == max(

matrix[x][y],

matrix[x - 1][y],

matrix[x][y - 1],

matrix[x][y + 1],

matrix[x + 1][y],

)

def find\_col\_peak(self, matrix, col, top, bottom):

peak\_val = -sys.maxsize

peak = None

for row in range(top, bottom + 1):

if matrix[row][col] > peak\_val:

peak\_val = matrix[row][col]

peak = row

return peak

def find\_row\_peak(self, matrix, row, left, right):

peak\_val = -sys.maxsize

peak = None

for col in range(left, right + 1):

if matrix[row][col] > peak\_val:

peak\_val = matrix[row][col]

peak = col

return peak

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

A = [

[1, 2, 3, 6, 5],

[16, 41, 23, 22, 6],

[15, 17, 24, 21, 7],

[14, 18, 19, 20, 10],

[13, 14, 11, 10, 9]

]

print('输入: ')

print(A)

print('输出: ')

print(temp.findPeakII(A))

例338 最长上升连续子序列

3.代码实现

class Solution:

#参数A为整数矩阵

#返回最长连续上升子序列的长度

def longestContinuousIncreasingSubsequence(self, A):

if not A or not A[0]:

return 0

n, m = len(A), len(A[0])

points = []

for i in range(n):

for j in range(m):

points.append((A[i][j], i, j))

points.sort()

longest\_hash = {}

for i in range(len(points)):

key = (points[i][1], points[i][2])

longest\_hash[key] = 1

for dx, dy in [(1, 0), (0, 1), (-1, 0), (0, -1)]:

x, y = points[i][1] + dx, points[i][2] + dy

if x < 0 or x >= n or y < 0 or y >= m:

continue

if (x, y) in longest\_hash and A[x][y] < points[i][0]:

longest\_hash[key] = max(longest\_hash[key], longest\_hash[(x, y)] + 1)

return max(longest\_hash.values())

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

A = [

[1, 2, 3, 4, 5],

[16, 17, 24, 23, 6],

[15, 18, 25, 22, 7],

[14, 19, 20, 21, 8],

[13, 12, 11, 10, 9]

]

print('输入: ')

print(A)

print('输出: ')

print(temp.longestContinuousIncreasingSubsequence(A))

例339 连续子数组求和

3.代码实现

import sys

class Solution:

#参数A为整数矩阵

#返回新数组起点位置和终点位置索引

def continuousSubarraySum(self, A):

max\_start, max\_end, max\_sum = self.find\_maximux\_subarray(A)

min\_start, min\_end, min\_sum = self.find\_maximux\_subarray([-a for a in A])

min\_sum = -min\_sum

total = sum(A)

if max\_sum > (total - min\_sum) or (min\_end - min\_start + 1) == len(A):

return [max\_start, max\_end]

if min\_start == 0:

return [min\_end + 1, len(A) - 1]

if min\_end == len(A) - 1:

return [0, min\_start - 1]

return [min\_end + 1, min\_start - 1]

def find\_maximux\_subarray(self, nums):

max\_sum = -sys.maxsize

curt\_sum, start = 0, 0

max\_range = []

for index, num in enumerate(nums):

if curt\_sum < 0:

curt\_sum = 0

start = index

curt\_sum += num

if curt\_sum > max\_sum:

max\_sum = curt\_sum

max\_range = [start, index]

return max\_range[0], max\_range[1], max\_sum-

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

A1 = [1, -1]

A2 = [3, 1, 2, -100, -3, -5, 4]

print('输入: ')

print(A1)

print('输出: ')

print(temp.continuousSubarraySum(A1))

print('输入: ')

print(A2)

print('输出: ')

print(temp.continuousSubarraySum(A2))

例340 子数组求和

3.代码实现

class Solution:

#参数A为一个整数数组

#参数start为给定区间范围左边界

#参数end为给定区间范围右边界

#返回子数组数量

def subarraySumII(self, A, start, end):

size = len(A)

sums = [0] \* (size + 1)

for i in range(size):

sums[i] = sums[i - 1] + A[i]

result = 0

for i in range(size):

for j in range(i, size):

if start <= sums[j] - sums[i - 1] <= end:

result += 1

return result

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

A = [2, 1, 4, 3]

start = 2

end = 6

print('输入: A=' + str(A) + ' start=' + str(start) + ' end=' + str(end))

print('输出: '+str(temp.subarraySumII(A, start, end)))

例341 找无向图的连通块

3.代码实现

class UndirectedGraphNode:

def \_\_init\_\_(self, x, neighbors=[]):

self.label = x

self.neighbors = neighbors

class Solution:

#参数nodes为无向图节点

#返回集合的列表

def connectedSet(self, nodes):

self.v = {}

for node in nodes:

self.v[node.label] = False

ret = []

for node in nodes:

if not self.v[node.label]:

tmp = []

self.dfs(node, tmp)

ret.append(sorted(tmp))

return ret

def dfs(self, node, tmp):

self.v[node.label] = True

tmp.append(node.label)

for node in node.neighbors:

if not self.v[node.label]:

self.dfs(node, tmp)

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

node1 = UndirectedGraphNode(1)

node3 = UndirectedGraphNode(3)

node2 = UndirectedGraphNode(2)

node4 = UndirectedGraphNode(4)

node5 = UndirectedGraphNode(5)

node1.neighbors = [node2, node4]

node2.neighbors = [node1, node4]

node4.neighbors = [node1, node2]

node3.neighbors = [node5]

node5.neighbors = [node3]

nodes = [node1, node2, node3, node4, node5]

print('输入: ')

print('{1,2,4#2,1,4#3,5#4,1,2#5,3}')

print('输出: ')

print(temp.connectedSet(nodes))

例342 硬币排成线

3.代码实现

class Solution:

#参数values为整数列表

#返回布尔型

def firstWillWin(self, values):

if not values:

return False

n = len(values)

dp = [[0] \* n for \_ in range(n)]

sum = [[0] \* n for \_ in range(n)]

for i in range(n):

dp[i][i] = values[i]

sum[i][i] = values[i]

for i in range(n - 2, -1, -1): #n-2 ≥ 0

for j in range(i + 1, n): #i+1 ≥ n-1

sum[i][j] = sum[i + 1][j] + values[i]

dp[i][j] = sum[i][j] - min(dp[i + 1][j], dp[i][j - 1])

return dp[0][n - 1] > sum[0][n - 1] - dp[0][n - 1]

if \_\_name\_\_=="\_\_main\_\_":

values=[1,2,4]

solution=Solution()

print("输入：",values)

print("输出：第一个玩家赢的情况是",solution.firstWillWin(values))

例343 检验IP地址

3.代码实现

class Solution(object):

def validIPAddress(self, IP):

#参数IP为字符串

#返回字符串

ip = IP.split('.')

if len(ip) == 4:

#ipv4

for octet\_s in ip:

try:

octet = int(octet\_s)

except ValueError:

return 'Neither'

if octet < 0 or octet > 255 or (octet\_s != '0' and (octet // 10\*\*(len(octet\_s) - 1) == 0)):

return 'Neither'

return 'IPv4'

else:

ip = IP.split(':')

if len(ip) == 8:

#ipv6

for hexa\_s in ip:

if not hexa\_s or len(hexa\_s) > 4 or not hexa\_s[0].isalnum():

return 'Neither'

try:

hexa = int(hexa\_s, base=16)

except ValueError:

return 'Neither'

hexa\_redo = '{:x}'.format(hexa)

if hexa < 0 or hexa > 65535:

return 'Neither'

return 'IPv6'

return 'Neither'

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n = '172.16.254.1'

print("输入：",n)

print("输出：",s.validIPAddress(n))

例344 环绕字符串中的唯一子串

3.代码实现

class Solution:

#参数p为字符串

#返回整数

def findSubstringInWraproundString(self, p):

res = {i: 1 for i in p}

l = 1

for i, j in zip(p, p[1:]):

l = l + 1 if (ord(j) - ord(i)) % 26 == 1 else 1

res[j] = max(res[j], l)

return sum(res.values())

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n = ['a']

print("输入：",n)

print("输出：",s.findSubstringInWraproundString(n))

例345 使数组元素相同的最少步数

3.代码实现

class Solution(object):

def minMoves2(self, nums):

#参数nums为数组

#返回整型

minmoves = 0

nums.sort()

median = nums[len(nums) // 2]

for i in range(0,len(nums)):

minmoves += abs(nums[i] - median)

return minmoves

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n = [1,2,3]

print("输入：",n)

print("输出：",s.minMoves2(n))

例346 重复的子串模式

3.代码实现

class Solution(object):

def repeatedSubstringPattern(self, s):

#参数s为字符串

#返回布尔类型

l = len(s)

next = [-1 for i in range(l)]

j = -1

for i in range(1, l):

while j >= 0 and s[i] != s[j + 1]:

j = next[j]

if s[i] == s[j + 1]:

j += 1

next[i] = j

lenSub = l - 1 - next[l - 1]

return lenSub != l and l % lenSub ==0

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n = 'abab'

print("输入：",n)

print("输出：",s.repeatedSubstringPattern(n))

例347 恢复二叉搜索树

3.代码实现

class TreeNode:

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

self.val = val

self.left, self.right = None, None

class Solution:

#参数root为二叉树根

#返回二叉树

def bstSwappedNode(self, root):

list\_val, list\_node = [], []

self.inorder(root, list\_val, list\_node)

list\_val.sort()

for i in range(len(list\_val)):

list\_node[i].val = list\_val[i]

return root

def inorder(self, node, list\_val, list\_node):

if not node: return

self.inorder(node.left, list\_val, list\_node)

list\_val.append(node.val)

list\_node.append(node)

self.inorder(node.right, list\_val, list\_node)

if \_\_name\_\_=='\_\_main\_\_':

solution=Solution()

root0=TreeNode(4)

root1=TreeNode(5)

root2=TreeNode(2)

root3=TreeNode(1)

root4=TreeNode(3)

root0.left=root1

root0.right=root2

root1.left=root3

root1.right=root4

print("输入:", root0.val,root0.left.val,root0.right.val,root0.left.left.val,root0.left.right.val)

temp=solution.bstSwappedNode(root0)

print("输出:", root0.val,root0.left.val,root0.right.val,root0.left.left.val,root0.left.right.val)

例348 数组中最大的差值

3.代码实现

class Solution:

#参数arrs为数组列表

#返回整数

def maxDiff(self, arrs):

length = len(arrs)

min\_val = arrs[0][0]

max\_val = arrs[0][-1]

max\_dif = 0

for i in range(1,length):

max\_dif = max(max(arrs[i][-1]-min\_val,max\_val-arrs[i][0]),max\_dif)

if arrs[i][0] < min\_val:

min\_val = arrs[i][0]

if arrs[i][-1] > max\_val:

max\_val = arrs[i][-1]

return max\_dif

if \_\_name\_\_=='\_\_main\_\_':

solution=Solution()

array=[[1,2,3], [4,5], [1,2,3]]

print("输入：",array)

print("输出：",solution.maxDiff(array))

例349 判断k个素数之和

3.代码实现

class Solution:

#参数n为整数

#参数k为整数

#返回布尔类型

def isSumOfKPrimes(self, n, k):

if n < 2\*k:

ret = False

elif k == 1:

ret = self.isPrime(n)

elif k == 2:

ret = (n % 2 == 0) or self.isPrime(n-2)

else:

ret = True

return ret

def isPrime(self, n):

i = 2

while i\*i <= n:

if n % i == 0:

return False

i += 1

return True

if \_\_name\_\_=='\_\_main\_\_':

solution=Solution()

n=10

k=2

print("输入：n=",n)

print("输入：k=",k)

print("输出：",solution.isSumOfKPrimes(n,k))

例350 杆子分割

3.代码实现

class Solution:

#参数prices为数组

#参数n为整数

#返回整数

def cutting(self, prices, n):

n = len(prices)

dp = [0]\*(n+1)

for i in range(1, n+1):

for j in range(i):

dp[i] = max(dp[i], dp[j]+prices[i - j - 1])

return dp[n]

if \_\_name\_\_=='\_\_main\_\_':

solution=Solution()

price=[1, 5, 8, 9, 10, 17, 17, 20]

n=8

print("输入价格：",price)

print("输入长度：",n)

print("输出最大值：",solution.cutting(price,n))

例351 二进制手表

3.代码实现

class Solution(object):

def readBinaryWatch(self, num):

#参数num为整数

#返回整数列表

ans = [];

hour = [[], [], [], []];

min = [[], [], [], [], [], []];

for i in range(0, 12):

n = bin(i).count('1')

hour[n].append(i)

for i in range(0, 60):

n = bin(i).count('1')

min[n].append(i)

for i in range(0, num + 1):

if i < 4 and num - i < 6:

for h in hour[i]:

for m in min[num - i]:

ans.append('%d:%02d' % (h, m));

return ans

if \_\_name\_\_=='\_\_main\_\_':

inputnum=1

solution=Solution()

print("输入：",inputnum)

print("输出：",solution.readBinaryWatch(inputnum))

例352 Data Segmentation

3.代码实现

class Solution:

#参数str为字符串

#返回字符串列表

def dataSegmentation(self, str):

ans = []

ins = ""

for c in str:

if c == ' ':

if ins != "":

ans.append(ins)

ins = ""

elif c >= 'a' and c <= 'z':

ins = ins + c

else:

if ins != "":

ans.append(ins)

ans.append(c)

ins = ""

if ins != "":

ans.append(ins)

return ans

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

s="(hi (i am)bye)"

print("输入：",s)

print("输出：",solution.dataSegmentation(s))

例353 位跟2位字符

3.代码实现

class Solution:

def isOneBitCharacter(self, bits):

"""

从头到尾遍历，如果该位数字为1，则向后前进两位，否则前进1位，循环的条件是i < n-1，即留出最后一位。

当循环退出后，i正好停留在n-1上，说明最后一位是单独分割开的。

"""

length = 0

while length < len(bits) - 1:

if bits[length]:

length += 2

else:

length += 1

return length == len(bits) - 1

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

num=[1,0,0]

print("输入：",num)

print("输出：",solution.isOneBitCharacter(num))

例354 加法数

3.代码实现

class Solution:

#参数num为字符串

#返回布尔类型

def isAdditiveNumber(self, num):

starting\_index = 0

prev\_two = []

num = str(num)

return self.dfs(num, starting\_index, prev\_two)

def dfs(self, num, starting\_index, prev\_two):

if starting\_index == len(num):

return False

if len(prev\_two) != 2:

for index in range(starting\_index + 1, len(num)):

prev\_two.append(int(num[starting\_index:index]))

if self.dfs(num, index, prev\_two):

prev\_two.pop()

return True

prev\_two.pop()

if num[starting\_index:index] == "0":

break

else:

sum\_needed = sum(prev\_two)

end\_index = len(str(sum\_needed)) + starting\_index

if num[starting\_index: end\_index] == str(sum\_needed):

if end\_index == len(num):

return True

if self.dfs(num, end\_index, [prev\_two[1], sum\_needed]):

return True

else:

return False

return False

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

num="112358"

print("输入：",num)

print("输出：",solution.isAdditiveNumber(num))

例355 具有交替位的二进制数

3.代码实现

class Solution:

#参数n为整数

#返回布尔类型

"""

一个具有交替位的二进制数，把它右移一位后与原二进制数异或，得到的新二进制数每一位上都是1。把问题转化为了如何判断一个二进制数是否全为1,采用的方法是将该二进制数+1后与原数进行操作。

"""

def hasAlternatingBits(self, n):

n = n ^ (n>>1)

return (n & n+1) == 0

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

n=5

print("输入：",n)

print("输出：",solution.hasAlternatingBits(5))

例356 美丽的排列

3.代码实现

class Solution:

#参数N为整数

#返回整数

def countArrangement(self, N):

cache = {}

return self.helper(cache, N, tuple(range(1, N + 1)))

def helper(self, cache, i, X):

if i == 1:

return 1

if (i, X) in cache:

return cache[(i, X)]

total = sum(self.helper(cache, i - 1, X[:j] + X[j + 1:]) for j, x in enumerate(X) if x % i == 0 or i % x == 0)

cache[(i, X)] = total

return total

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

num=2

print("输入：",num)

print("输出：",solution.countArrangement(num))

例357 最大值在界内的子数组个数

3.代码实现

class Solution:

#参数A为数组

#参数L为整数

#参数R为整数

#返回整数

def numSubarrayBoundedMax(self, A, L, R):

n = len(A)

if n == 0:

return 0

ans = 0

l, r = 0, 0

while l < n:

maxv = A[r]

while (maxv <= R): #以大于R的元素为分割

r += 1

if r >= n:

break

maxv = max(maxv, A[r])

#当前[l, r)范围的下标构成子数组是被大于R的元素分割出来的一段

ans += (r - l) \* (r - l + 1) // 2 #这一段的全体子数组的数量

last = l - 1 #last表示上一个不小于L的元素位置

while l < r: #运用容斥原理把这一段内非法的再减去

if (A[l] >= L):

ans -= (l - last) \* (l - last - 1) // 2

last = l;

l += 1

ans -= (r - last) \* (r - last - 1) // 2 #不要忘记末尾还有一段

l += 1

r += 1

return ans

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

L=2

R=3

arr=[2, 1, 4, 3]

print("输入L：",L)

print("输入R：",R)

print("输入数组：",arr)

print("输出数量：",solution.numSubarrayBoundedMax(arr,L,R))

例358 全局和局部逆序数

3.代码实现

class Solution:

def isIdealPermutation(self, A):

#参数A为整型列表

#返回值为布尔型

max\_value = 0

for i in range(2, len(A)):

max\_value = max(A[i - 2], max\_value)

if max\_value > A[i]:

return False

return True

if \_\_name\_\_ == '\_\_main\_\_':

solution=Solution()

num=[1,0,2]

print("输入：",num)

print("输出：",solution.isIdealPermutation(num))

例359 整数拆分

3.代码实现

class Solution:

#参数n为整数

#返回整数

def integerBreak(self, n):

dp = [0] \* (1 + n)

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

for i in range(1, j):

dp[j] = max(dp[j], max(i, dp[i])\*max(j - i, dp[j - i]))

return dp[n]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

Test\_in = 10

Test\_out = solution.integerBreak(Test\_in)

print("输入：",Test\_in)

print("输出：",Test\_out)

例360 递增的三元子序列

3.代码实现

class Solution:

#参数nums为整数数组

#返回布尔型

def increasingTriplet(self, nums):

if len(nums) < 3:

return False

import sys

first = sys.maxsize

second = sys.maxsize

for num in nums:

if num <= first:

sec = first

first = num

elif num <= sec:

sec = num

else:

return True

return False

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

Test\_in = [1,2,3,4,5]

Test\_out = solution.increasingTriplet(Test\_in)

print("输入：",Test\_in)

print("输出：",Test\_out)

例361 重新安排行程

3.代码实现

import collections

class Solution:

#tickets为字符串列表

#返回字符串列表

def findItinerary(self, tickets):

targets = collections.defaultdict(list)

for a, b in sorted(tickets)[::-1]:

targets[a] += b,

route = []

def dfs(airport):

while targets[airport]:

dfs(targets[airport].pop())

route.append(airport)

dfs('JFK')

return route[::-1]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

Test\_in = [["MUC", "LHR"], ["JFK", "MUC"], ["SFO", "SJC"], ["LHR", "SFO"]]

Test\_out = solution.findItinerary(Test\_in)

print("输入：",Test\_in)

print("输出：",Test\_out)

例362 奇偶链表

3.代码实现

class ListNode(object):

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

self.val = val

self.next = next

class Solution:

#参数head为链表头

#返回修改的链表

def oddEvenList(self, head):

if head is None:

return head

odd = head

evenHead = head.next

even = evenHead

while even and even.next:

odd.next = even.next

odd = odd.next

even.next = odd.next

even = even.next

odd.next = evenHead

return head

def getLinkedList(head):

list = []

while head is not None:

list += [head.val]

head = head.next

list.append('null')

s = '->'.join(list)

return s

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

LinkedNode1 = ListNode('1', ListNode('2', ListNode('3', ListNode('4', ListNode('5', ListNode('6'))))))

print('输入: ' + getLinkedList(LinkedNode1))

print('输出: ' + getLinkedList(solution.oddEvenList(LinkedNode1)))

例363 区间和的个数

3.代码实现

class Solution:

def countRangeSum(self, nums, lower, upper):

n = len(nums)

count\_sum = {0:1}

pre\_sum = 0

ans = 0

for num in nums:

pre\_sum += num

for d in range(lower, upper + 1):

if pre\_sum - d in count\_sum:

ans += count\_sum[pre\_sum - d]

count\_sum[pre\_sum] = count\_sum.get(pre\_sum, 0) + 1

return ans

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

nums = [-2, 5, -1]

lower = -2

upper = 2

Test\_out = solution.countRangeSum(nums, lower, upper)

print("输入数组：",nums)

print("输入下限：",lower)

print("输出上限：",upper)

print("输出结果：",Test\_out)

例364 3的幂

3.代码实现

class Solution:

#参数n为整数

#返回布尔类型

def isPowerOfThree(self, n):

n = abs(n)

num = 1

while num <= n:

if num == n:

return True

num \*= 3

return False

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

Test\_in = 9

Test\_out = solution.isPowerOfThree(Test\_in)

print("输入：",Test\_in)

print("输出：",Test\_out)

例365 单词长度最大积

3.代码实现

from collections import Counter

class Solution(object):

def maxProduct(self, words):

#参数words为字符串列表

#返回整数

sets = Counter()

for w in words:

key = frozenset(w)

sets[key] = max(sets[key], len(w))

max\_len = 0

for x, vx in sets.items():

for y, vy in sets.items():

if not x.intersection(y):

max\_len = max(max\_len, vx \* vy)

return max\_len

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

Test\_in = ["abcw", "baz", "foo", "bar", "xtfn", "abcdef"]

Test\_out = solution.maxProduct(Test\_in)

print("输入：",Test\_in)

print("输出：",Test\_out)

例366 矩阵注水

3.代码实现

class Solution:

vis = []

def bfs(self, matrix, R, C):

if R == 0 or R == (len(matrix) - 1) or C == 0 or C == (len(matrix[0]) - 1) :

return True

self.vis[R][C] = 1

if (matrix[R][C+1] - matrix[R][C]) < 0:

if self.bfs(matrix, R, C+1):

return True

if (matrix[R][C-1] - matrix[R][C]) < 0:

if self.bfs(matrix, R, C-1):

return True

if (matrix[R+1][C] - matrix[R][C]) < 0:

if self.bfs(matrix, R+1, C):

return True

if (matrix[R-1][C] - matrix[R][C]) < 0:

if self.bfs(matrix, R-1, C):

return True

return False

def waterInjection(self, matrix, R, C):

if R == 0 or R == (len(matrix) - 1) or C == 0 or C == (len(matrix[0]) - 1) :

return "YES"

self.vis = [[0 for i in range(len(matrix[0]))] for j in range(len(matrix))]

self.vis[R][C] = 1

if (matrix[R][C + 1] - matrix[R][C]) < 0:

if self.bfs(matrix, R, C + 1):

return "YES"

if (matrix[R][C - 1] - matrix[R][C]) < 0:

if self.bfs(matrix, R, C - 1):

return "YES"

if (matrix[R + 1][C] - matrix[R][C]) < 0:

if self.bfs(matrix, R + 1, C):

return "YES"

if (matrix[R - 1][C] - matrix[R][C]) < 0:

if self.bfs(matrix, R - 1, C):

return "YES"

return "NO"

if \_\_name\_\_ == '\_\_main\_\_':

R = 1

C = 1

matrix = [[10,18,13],[9,8,7],[1,2,3]]

solution = Solution()

print(" R与C分别为：", R, C)

print(" 矩阵：", matrix)

print(" 是否流出:", solution.waterInjection(matrix, R, C))

例367 最大值

3.代码实现

class Solution:

def pickThreeNumbers(self, p, q, r, a):

f = [[0 for i in range(len(a))] for i in range(3)]

fac = [p,q,r]

f[0][0] = a[0] \* p

f[1][0] = f[0][0] + a[0] \* q

f[2][0] = f[1][0] + a[0] \* r

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

f[0][i] = max(f[0][i-1], a[i] \* p)

for i in range(1, 3) :

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

f[i][j] = max(f[i-1][j] + a[j] \* fac[i], f[i][j-1])

return f[2][len(a) - 1]

if \_\_name\_\_ == '\_\_main\_\_':

p,q,r = -1,-2,3

a = [-1,2,-3,4,5]

solution = Solution()

print("输入p,q,r分别为:", p,q,r)

print("输出最大计算结果:", solution.pickThreeNumbers(p, q, r, a))

例368 吃豆子

3.代码实现

class Solution:

def eatTheBeans(self, w, r):

n = 2 \* (w + r)

dp = [[[0 for k in range(r + 2)] for j in range(w + 2)] for i in range(n)]

dp[0][w][r] = 1.0

for i in range(0, n - 1):

for j in range(w + 1):

for k in range(r + 1):

if dp[i][j][k] > 0:

if j == 0 and k == 0:

continue

if j == 0:

if i % 2 == 1:

dp[i + 1][0][k - 1] += dp[i][j][k]

else:

dp[i + 1][0][k] += dp[i][j][k]

elif k == 0:

dp[i + 1][j - 1][0] += dp[i][j][k]

else:

if i % 2 == 1:

dp[i + 1][j - 1][k] += 1.0 \* j / (j + k) \* dp[i][j][k]

dp[i + 1][j][k - 1] += 1.0 \* k / (j + k) \* dp[i][j][k]

else:

dp[i + 1][j - 1][k] += 1.0 \* j / (j + k) \* dp[i][j][k]

dp[i + 1][j][k] += 1.0 \* k / (j + k) \* dp[i][j][k]

ans = 0

for i in range(n):

ans += dp[i][1][0]

return 1.0 \* ans

if \_\_name\_\_ == '\_\_main\_\_':

w,r = 1,1

solution = Solution()

print(" 白豆子和红豆子数量：", w,r)

print(" 最后一个是白豆子的概率:", solution.eatTheBeans(w, r))

例369 余积

3.代码实现

class Solution:

def getProduct(self, arr):

n = len(arr)

ans = [0 for i in range(n)]

if n == 1:

ans[0] = 0

return ans

flag = 0

for i in range(n): #搜索0的个数

if arr[i] == 0:

flag += 1

if flag == 0:

p = 1

for i in range(n):

p \*= arr[i]

for i in range(n):

ans[i] = p / arr[i]

elif flag == 1:

p = 1

for i in range(n):

if arr[i] != 0:

p \*= arr[i]

for i in range(n):

if arr[i] == 0:

ans[i] = p

else:

ans[i] = 0

else:

for i in range(n):

ans[i] = 0

return ans

if \_\_name\_\_ == '\_\_main\_\_':

arr = [1,2,3,4,5,6,0]

solution = Solution()

print(" 输入序列：", arr)

print(" 结果序列:", solution.getProduct(arr))

例370 前一个数

3.代码实现

class Solution:

def getPreviousNumber(self, num):

stk = []

n = len(num)

ans = [0 for i in range(n)]

for i in range(n):

while len(stk) > 0 and stk[-1] >= num[i]:

stk.pop()

if len(stk) > 0:

ans[i] = stk[-1]

else:

ans[i] = num[i]

stk.append(num[i]) #push

return ans

if \_\_name\_\_ == '\_\_main\_\_':

num = [6,3,1,2,5,10,9,15]

solution = Solution()

print(" 输入序列：", num)

print(" 结果序列：", solution.getPreviousNumber(num))

例371 称重问题

3.代码实现

class Solution:

def minimumtimes(self, n):

ans = 0

if n % 3 ==0:

n -= 1

while not n <= 1:

n /= 3

ans += 1

return ans

if \_\_name\_\_ == '\_\_main\_\_':

n = 4

solution = Solution()

print(" 金币总数：", n)

print(" 最少需要称重次数：", solution.minimumtimes(n))

例372 树中距离的总和

3.代码实现

class Solution:

def sumOfDistancesInTree(self, N, edges):

distances = [0] \* N

counts = [1] \* N

neighbors = [[] for \_ in range(N)]

for e in edges:

neighbors[e[0]].append(e[1])

neighbors[e[1]].append(e[0])

def dfsCountNode(parent, node):

for nei in neighbors[node]:

if nei == parent:

continue

dfsCountNode(node, nei)

counts[node] += counts[nei]

distances[node] += distances[nei] + counts[nei]

def dfs2(parent, node):

if parent != -1:

distances[node] = distances[parent] - counts[node] + N - counts[node]

for nei in neighbors[node]:

if nei == parent:

continue

dfs2(node, nei)

dfsCountNode(-1, 0)

dfs2(-1, 0)

return distances

if \_\_name\_\_ == '\_\_main\_\_':

N = 6

edges = [[0,1],[0,2],[2,3],[2,4],[2,5]]

solution = Solution()

print(" 总共有", N,"个节点，边对列表为：",edges)

print(" 结果序列:", solution.sumOfDistancesInTree(N, edges))

例373 订单问题

3.代码实现

class Solution:

#参数order为需求订单

#参数pattern为生产模式的二维列表

#返回值为整数，是剩余需求的和

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

self.ans = 100000

def dfs(self,now,order):#order：剩余需求量的列表

if now == self.m:

sum = 0

for i in range(0,self.n):

sum += order[i]

self.ans=min(sum, self.ans)

return

flag = 0

for i in range(0, self.n):

if self.arr[now][i] > 0:#arr同于pattern

flag = 1

tmp=[]

for i in range(0,self.n):

tmp.append(order[i])

if flag==0:

self.dfs(now + 1, tmp)

return

self.dfs(now + 1, tmp)

while True:

flag = 0

for i in range(0,self.n):

order[i] -= self.arr[now][i]

for i in range(0, self.n):

if order[i] < 0:

flag = 1

break

if flag == 1:

break

for i in range(0, self.n):

tmp[i]=(order[i])

self.dfs(now + 1, tmp)

def getMinRemaining(self, order, pattern):

self.n = len(order)

self.m = len(pattern)

self.arr = pattern

tmp =order

self.dfs(0, tmp)

return self.ans

if \_\_name\_\_ == '\_\_main\_\_':

order=[2,3,1]

pattern=[[2,2,0],[0,1,1],[1,1,0]]

solution = Solution()

print(" 有订单：", order,"有生产模式的列表：",pattern)

print(" 剩余需求：", solution.getMinRemaining(order, pattern))

例374 LFU缓存

3.代码实现

from collections import OrderedDict

from collections import defaultdict

class LFUCache:

#参数capacity为整数

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

self.mincount = 0

self.capacity = capacity

self.cache = {}

self.visited = {}

#默认字典嵌套有序字典，外层字典的键是访问次数，有序字典根据放入元素的先后顺序进行排序

self.key\_list = defaultdict(OrderedDict)

#参数key为整数

#参数value为整数

#无返回值

def set(self, key, value):

#如果该key已经存在，修改value并且次数+1

if key in self.cache:

self.cache[key] = value

self.get(key)

return

#如果缓存已满，则删除最少访问次数

if len(self.cache) == self.capacity:

#找到最小访问次数

temp\_key, tmep\_val = next(iter(self.key\_list[self.mincount].items()))

#min\_visit = min(self.visited, key=lambda x: self.visited[x])

del self.cache[temp\_key]

del self.visited[temp\_key]

del self.key\_list[self.mincount][temp\_key]

self.cache[key] = value

self.visited[key] = 0

#添加时默认是1，所以都放在访问次数为1的层中

self.mincount = 1

self.cache[key] = value

self.visited[key] = 1

#对记录字典进行赋值{1：{key:none, key1:none}}

self.key\_list[1][key] = None

#参数key为整数

#返回值为整数

def get(self, key):

if key not in self.cache:

return -1

#取出该key的访问次数

count = self.visited[key]

#对访问次数进行+1

self.visited[key] += 1

#对记录字典进行更新

self.key\_list[count].pop(key)

self.key\_list[count+1][key] = None

#等于最小访问次数，并且该次数已经没有值，则最小访问次数+1，为下次加入做准备

if count == self.mincount and len(self.key\_list[count]) == 0:

self.mincount += 1

return self.cache[key]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

LFU = LFUCache(3)

LFU.set(2, 2)

LFU.set(1, 1)

res1 = LFU.get(2)

res2 = LFU.get(1)

res3 = LFU.get(2)

LFU.set(3, 3)

LFU.set(4, 4)

res4 = LFU.get(3)

res5 = LFU.get(2)

res6 = LFU.get(1)

res7 = LFU.get(4)

a = [res1, res2, res3, res4, res5, res6, res7]

print("输入：\n LFUCache(3) \n set(2, 2) \n set(1, 1) \n get(2) \n get(1) \n get(2) \n set(3, 3) \n set(4, 4) \n get(3) \n get(2) \n get(1) \n get(4)")

print("输出：", a)

例375 音乐播放表

3.代码实现

#参数n表示歌单里有n首歌

#参数m表示两首相同歌之间至少间隔m 首歌

#参数p总共能听p首歌

#返回值为整数，为方案的总和

mod = 1000000007

class Solution:

def getAns(self, n, m, p):

dp = [[0 for i in range(n+1)] for j in range(p+1)]

for i in range(p+1):

for j in range(n+1):

dp[i][j] = 0

dp[0][0] = 1;

for i in range(1, p+1):

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

dp[i][j] += dp[i-1][j-1] \* (n - j + 1);

dp[i][j] %= mod;

if (j > m):

dp[i][j] += dp[i-1][j] \* (j - m)

dp[i][j] %= mod

return dp[p][n]

if \_\_name\_\_ == '\_\_main\_\_':

n = 2

m = 0

p = 3

solution = Solution()

print(" 歌单里有", n, "首歌，两首相同歌之间至少间隔",m , "首歌，总共能听",p ,"首歌")

print(" 可以生成：", solution.getAns(n, m, p), "个歌单")

例376 阶乘

3.代码实现

import math

class Solution:

#参数n为整数

#返回字符串

def factorial(self, n):

return str(math.factorial(n))

if \_\_name\_\_=='\_\_main\_\_':

solution=Solution()

n=20

print("输入：",n)

print("输出：",solution.factorial(n))

例377 解码方式

3.代码实现

class Solution(object):

def numDecodings(self, s):

#参数s为字符串

#返回整数

if s == None:

return 0

mod = 1000000007

n = len(s)

f = [0] \* (n + 1)

f[0] = 1

for i in range(1, n + 1):

if s[i - 1] == '\*':

f[i] = (f[i] + 9 \* f[i - 1]) % mod

if i >= 2:

t = 0

if s[i - 2] == '\*':

f[i] = (f[i] + 15 \* f[i - 2]) % mod

elif s[i - 2] == '1':

f[i] = (f[i] + 9 \* f[i - 2]) % mod

elif s[i - 2] == '2':

f[i] = (f[i] + 6 \* f[i - 2]) % mod

else:

if s[i - 1] >= '1' and s[i - 1] <= '9':

f[i] = (f[i] + f[i - 1]) % mod

if i >= 2:

if s[i - 2] == '\*':

t = 0

if s[i - 1] >= '0' and s[i - 1] <= '6':

f[i] = (f[i] + 2 \* f[i - 2]) % mod

elif s[i - 1] >= '7' and s[i - 1] <= '9':

f[i] = (f[i] + f[i - 2]) % mod

else:

twoDigits = int(s[i - 2 : i])

if twoDigits >= 10 and twoDigits <= 26:

f[i] = (f[i] + f[i - 2]) % mod

return f[n]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

s = "1\*"

ans = solution.numDecodings(s)

print("输入：", s)

print("输出：", ans)

例378 最大树

3.代码实现

#定义树节点

class TreeNode:

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

self.val = val

self.left, self.right = None, None

class Solution:

#参数A: 给定没有重复项的整数数组

#返回值: 最大树的根

def maxTree(self, A):

stack = []

for element in A:

node = TreeNode(element)

while len(stack) != 0 and element > stack[-1].val:

node.left = stack.pop()

if len(stack) != 0:

stack[-1].right = node

stack.append(node)

return stack[0]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

A = [2,5,6,0,3,1]

tree = solution.maxTree(A)

t = [tree.val, tree.left.val, tree.right.val, tree.left.left.val,

tree.right.left.val, tree.right.right.val]

print("输入：", A)

print("输出：", t)

例379 单词搜索

3.代码实现

DIRECTIONS = [(0, -1), (0, 1), (-1, 0), (1, 0)]

class TrieNode: #定义字典树的节点

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

self.children = {}

self.is\_word = False

self.word = None

class Trie:

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

self.root = TrieNode()

def add(self, word): #字典树插入单词

node = self.root

for c in word:

if c not in node.children:

node.children[c] = TrieNode()

node = node.children[c] #继续遍历

node.is\_word = True

node.word = word #存入单词

def find(self, word):

node = self.root

for c in word:

node = node.children.get(c)

if node is None:

return None

return node

class Solution:

#参数board: 二维字符串列表

#参数words: 字符串列表

#返回值: 字符串列表

def wordSearchII(self, board, words):

if board is None or len(board) == 0:

return []

trie = Trie()

for word in words: #插入单词

trie.add(word)

result = set()

for i in range(len(board)):#遍历字母矩阵，将每个字母作为单词首字母开始搜索

for j in range(len(board[0])):

c = board[i][j]

self.search(

board,

i,

j,

trie.root.children.get(c),

set([(i, j)]),

result,

)

return list(result)

def search(self, board, x, y, node, visited, result):#在字典树上dfs查找

if node is None:

return

if node.is\_word:

result.add(node.word)

for delta\_x, delta\_y in DIRECTIONS: #向四个方向查找

x\_ = x + delta\_x

y\_ = y + delta\_y

if not self.inside(board, x\_, y\_):

continue

if (x\_, y\_) in visited:

continue

visited.add((x\_, y\_))

self.search(

board,

x\_,

y\_,

node.children.get(board[x\_][y\_]),

visited,

result,

)

visited.remove((x\_, y\_))

def inside(self, board, x, y):

return 0 <= x < len(board) and 0 <= y < len(board[0])

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

board = [list("doaf"),list("agai"),list("dcan")]

words = ["dog","dad","dgdg","can","again"]

result = solution.wordSearchII(board, words)

print('输入：["doaf","agai","dcan"], ["dog","dad","dgdg","can","again"]')

print("输出：", result)

例380 LRU缓存策略

3.代码实现

class LinkedNode:

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

self.key = key

self.value = value

self.next = next

class LRUCache:

#参数capacity为整数

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

self.hash = {}

self.head = LinkedNode()

self.tail = self.head

self.capacity = capacity

def push\_back(self, node):

self.hash[node.key] = self.tail

self.tail.next = node

self.tail = node

def pop\_front(self): #删除头部

del self.hash[self.head.next.key]

self.head.next = self.head.next.next

self.hash[self.head.next.key] = self.head

def kick(self, prev): #将数据移动至尾部

node = prev.next

if node == self.tail:

return

prev.next = node.next

if node.next is not None:

self.hash[node.next.key] = prev

node.next = None

self.push\_back(node)

def get(self, key): #获取数据

if key not in self.hash:

return -1

self.kick(self.hash[key])

return self.hash[key].next.value

#参数key为整数

#参数value为整数

#返回值：无

def set(self, key, value): #数据放入缓存

if key in self.hash:

self.kick(self.hash[key])

self.hash[key].next.value = value

else:

self.push\_back(LinkedNode(key, value)) #如果key不存在，则存入新节点

if len(self.hash) > self.capacity: #如果缓存超出上限

self.pop\_front() #删除头部

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

lru = LRUCache(2)

lru.set(2, 1)

lru.set(1, 1)

ans1 = lru.get(2)

lru.set(4, 1)

ans2 = lru.get(1)

ans3 = lru.get(2)

a = [ans1, ans2, ans3]

if a == [1, -1, 1]:

print("输入：LRUCache(2), set(2, 1), set(1, 1), get(2), set(4, 1), get(1), get(2)")

print("输出：", a)

例381 书籍复印

3.代码实现

#参数pages是个整型数数组

#参数k是一个整数，表示几个人复印书籍

#返回值是一个整型数表示最少需要的时间

class Solution:

def copyBooks(self, pages, k):

if not pages:

return 0

start, end = max(pages), sum(pages)

while start + 1 < end:

mid = (start + end) // 2

if self.get\_least\_people(pages, mid) <= k:

end = mid

else:

start = mid

if self.get\_least\_people(pages, start) <= k:

return start

return end

def get\_least\_people(self, pages, time\_limit):

count = 0

time\_cost = 0

for page in pages:

if time\_cost + page > time\_limit:

count += 1

time\_cost = 0

time\_cost += page

return count + 1

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generator=[3,2,4]

k=2

solution = Solution()

print("输入：", generator)

print("输出：", solution. copyBooks(generator,k))

例382 二进制表示

3.代码实现

from decimal import \*

class Solution:

#参数n为给定作为字符串传入的十进制数

#返回值为字符串

def binaryRepresentation(self, num):

(a, b) = num.split('.')

a = '{:b}'.format(int(a))

b = self.frac\_to\_binary(b)

if b is None:

return 'ERROR'

elif b == '':

return a

else:

return a + '.' + b

def frac\_to\_binary(self, num):

if int(num) == 0:

return ''

if int(num) % 10 != 5:

return None

res = ''

num = Decimal('0.' + str(num))

while num:

num \*= 2

if num >= 1:

res += '1'

num -= 1

else:

res += '0'

num = num.normalize()

if num and str(num)[-1] != '5':

return None

return res

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

num = 3.5

ans = solution.binaryRepresentation(str(num))

print("输入：", num)

print("输出：", ans)

例383 房屋染色

3.代码实现

#参数costs是一个n行k列 表示花费的矩阵

#返回值res是染色的最小总花费

import sys

class Solution:

def minCostII(self, costs):

n = len(costs)

if not n:return 0

k = len(costs[0])

if not k:return 0

f = [[0]\*k for i in range(n+1)]

min1,min2 = 0,0

id1,id2 =0,0

for i in range(1,n+1):

min1 = min2 = sys.maxsize

for j in range(k):

if f[i-1][j]<min1:

min2 = min1

min1 = f[i-1][j]

id1 = j

else:

if f[i-1][j] < min2:

min2 = f[i-1][j]

id2 = j

for j in range(k):

f[i][j] = costs[i-1][j]

if j != id1:

f[i][j]+=min1

else:

f[i][j]+=min2

res = min(f[-1])

return res

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generation=[[14,2,11],[11,14,5],[14,3,10]]

solution= Solution()

print("输入：", generation)

print("输出：", solution.minCostII(generation))

例384 数组最大价值

3.代码实现

#参数a与b分别是匹配数组和价值数组

#返回值是整数，代表选择区间的最大价值

class Solution:

def getAnswer(self, a, b):

ans = [-1 for i in range(len(a))]

ans[0] = 0

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

for j in range(i):

if a[i] == a[j]:

temp = 0

for k in range(j, i + 1):

temp += b[k]

ans[i] = max(a[i - 1], temp)

break

if ans[i] == -1:

ans[i] = ans[i - 1]

return ans[len(a) - 1]

if \_\_name\_\_ == '\_\_main\_\_':

a = [1,2,3,4,2,6]

b = [1,2,1,2,1,100]

solution = Solution()

print("输入a与b：", a, b)

print("输出最大价值:", solution.getAnswer(a, b))

例385 最大字数组

3.代码实现

class Solution:

#参数nums为整型数组

#参数k1为整数

#参数k2为整数

#返回最大和

def maxSubarray5(self, nums, k1, k2):

n = len(nums)

if n < k1:

return 0

import sys

result = -sys.maxsize

sum = [0 for \_ in range(n + 1)]

from collections import deque

queue = deque()

for i in range(1, n + 1):

sum[i] = sum[i - 1] + nums[i - 1]

if len(queue) and queue[0] < i- k2:

queue.popleft()

if i >= k1:

while len(queue) and sum[queue[-1]] > sum[i - k1]:

queue.pop()

queue.append(i - k1)

if len(queue) and sum[i] - sum[queue[0]] > result:

result = max(result, sum[i] - sum[queue[0]])

return result

#主函数

if \_\_name\_\_=='\_\_main\_\_':

inputnum=[-2,2,-3,4,-1,2,1,-5,3]

k1 = 2

k2 = 4

solution=Solution()

print("输入数组：",inputnum)

print("输入k1=：",k1)

print("输入k2=：",k2)

print("输出sum=：",solution.maxSubarray5(inputnum,k1,k2))

例386 青蛙跳

3.代码实现

class Solution:

#参数stones 是石头位置列表

#返回青蛙是否可以过河

def canCross(self, stones):

dp = {}

for stone in stones:

dp[stone] = set([])

dp[0].add(0)

for stone in stones:

for k in dp[stone]:

#k - 1

if k - 1 > 0 and stone + k - 1 in dp:

dp[stone + k - 1].add(k - 1)

#k

if stone + k in dp:

dp[stone + k].add(k)

#k + 1

if stone + k + 1 in dp:

dp[stone + k + 1].add(k + 1)

return len(dp[stones[-1]]) > 0

#主函数

if \_\_name\_\_=='\_\_main\_\_':

inputnum=[0,1,3,5,6,8,12,17]

solution=Solution()

print("输入：",inputnum)

print("输出：",solution.canCross(inputnum))

例387 二叉搜索树中最接近的值

3.代码实现

class TreeNode:

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

self.val = val

self.left, self.right = None, None

class Solution:

#参数root为二叉搜索树

#参数target为给定目标值

#参数k为整数

#返回k个数

def closestKValues(self, root, target, k):

if root is None or k == 0:

return []

lower\_stack = self.get\_stack(root, target)

upper\_stack = list(lower\_stack)

if lower\_stack[-1].val < target:

self.move\_upper(upper\_stack)

else:

self.move\_lower(lower\_stack)

result = []

for i in range(k):

if self.is\_lower\_closer(lower\_stack, upper\_stack, target):

result.append(lower\_stack[-1].val)

self.move\_lower(lower\_stack)

else:

result.append(upper\_stack[-1].val)

self.move\_upper(upper\_stack)

return result

def get\_stack(self, root, target):

stack = []

while root:

stack.append(root)

if target < root.val:

root = root.left

else:

root = root.right

return stack

def move\_upper(self, stack):

if stack[-1].right:

node = stack[-1].right

while node:

stack.append(node)

node = node.left

else:

node = stack.pop()

while stack and stack[-1].right == node:

node = stack.pop()

def move\_lower(self, stack):

if stack[-1].left:

node = stack[-1].left

while node:

stack.append(node)

node = node.right

else:

node = stack.pop()

while stack and stack[-1].left == node:

node = stack.pop()

def is\_lower\_closer(self, lower\_stack, upper\_stack, target):

if not lower\_stack:

return False

if not upper\_stack:

return True

return target - lower\_stack[-1].val < upper\_stack[-1].val - target

if \_\_name\_\_ == '\_\_main\_\_':

root = TreeNode(1)

target = 0.000000

k = 1

solution = Solution()

print("输入k：",k)

print("输出：",solution.closestKValues(root, target, k))

例388 k步编辑

3.代码实现

class TrieNode:

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

#定义数据结构

self.children = [None for i in range(26)]

self.hasWord = False

self.str = None

@classmethod

def addWord(cls, root, word):

node = root

for letter in word:

child = node.children[ord(letter) - ord('a')]

if child is None:

child = TrieNode()

node.children[ord(letter) - ord('a')] = child

node = child

node.hasWord = True

node.str = word

class Solution:

#参数words为一组字符串

#参数target为目标字符串

#参数k为整数

#返回满足需求的字符串

def kDistance(self, words, target, k):

root = TrieNode()

for word in words:

TrieNode.addWord(root, word)

result = []

n = len(target)

dp = [i for i in range(n + 1)]

self.find(root, result, k, target, dp)

return result

def find(self, node, result, k, target, dp):

n = len(target)

if node.hasWord and dp[n] <= k:

result.append(node.str)

next = [0 for i in range(n + 1)]

for i in range(26):

if node.children[i] is not None:

next[0] = dp[0] + 1

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

if ord(target[j - 1]) - ord('a') == i:

next[j]= min(dp[j - 1], min(next[j - 1] + 1, dp[j] + 1))

else:

next[j]=min(dp[j-1]+1, min(next[j-1]+1, dp[j] + 1))

self.find(node.children[i], result, k, target, next)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

inputwords=["abc","abd","abcd","adc"]

target="ac"

k=1

solution=Solution()

print("输入字符串：",inputwords)

print("目标字符串：",target)

print("输出字符串：",solution.kDistance(inputwords,target,k))

例389 符号串生成器

3.代码实现

class Solution:

#参数S为字符串

#参数K为整数

#返回字符

def licenseKeyFormatting(self, S, K):

S = S.replace("-", "").upper()

count = 0

res = ""

for c in S[::-1]:

if count == K:

res = "-" + res

count = 0

res = c + res

count += 1

return res

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

S = '5F3Z-2e-9-w'

K = 4

print("输入字符串：",S)

print("输入分组长度：",K)

print("输出：",s.licenseKeyFormatting(S,K))

例390 单词合成问题

3.代码实现

#参数target代表目标被组合的单词

#参数words是可选单词集合

#返回值是一个布尔值，代表能否被合成这个目标单词

class Solution:

arr = [[] for i in range(20)]

belong = [-1 for i in range(20)]

vis = [False for i in range(20)]

def dfs(self, now):

for i in range(len(self.arr[now])):

v = self.arr[now][i]

if self.vis[v]:

continue

self.vis[v] = True

if self.belong[v] == -1 or self.dfs(self.belong[v]):

self.belong[v] = now

return True

return False

def matchFunction(self, target, words):

lent = len(target)

for i in range(lent):

for j in range(len(words)):

if target[i] in words[j]:

self.arr[i].append(j)

for i in range(lent):

self.vis = [False for i in range(20)]

if not self.dfs(i):

return False

return True

if \_\_name\_\_ == '\_\_main\_\_':

target="ray"

words=["buy","discard","lip","rep"]

solution = Solution()

print(" 目标：", target)

print(" 单词组：", words)

print(" 能否组成:", solution.matchFunction(target, words))

例391 最长数列

3.代码实现

#参数a为输入原始数组

#返回值是整数，代表最大数组长度

class Solution:

def getAnswer(self, a):

dp = [[2 for i in range(5050)] for j in range(5050)]

n = len(a)

a.sort();

ans = 2

for i in range(0, n):

l = i - 1;

r = i + 1

while (l >= 0 and r < n):

if (a[l] + a[r] == a[i] \* 2):

dp[i][r] = max(dp[i][r], dp[l][i] + 1)

ans = max(ans, dp[i][r])

r += 1;

l -= 1;

elif (a[l] + a[r] < a[i] \* 2):

r += 1;

else:

l -= 1;

return ans

if \_\_name\_\_ == '\_\_main\_\_':

a = [1,2,5,9,10]

solution = Solution()

print(" 输入数组：", a)

print(" 最大长度：", solution.getAnswer(a))

例392 拆分子数组

3.代码实现

class Solution:

#参数nums为整数列表

#参数m为整数

#返回整数

def splitArray(self, nums, m):

if not nums:

return 0

n = len(nums)

start = max(nums)

end = sum(nums)

while start + 1 < end:

largest\_sum = (start + end) // 2

if self.largest\_sum\_satisfy\_m( nums, m, largest\_sum ):

end = largest\_sum

else:

start = largest\_sum

if self.largest\_sum\_satisfy\_m( nums, m, start):

return start

return end

def largest\_sum\_satisfy\_m(self, nums, m, largest\_sum):

num\_of\_sub = 0

curr\_sum = 0

for num in nums:

if curr\_sum + num <= largest\_sum:

curr\_sum += num

else:

num\_of\_sub += 1

curr\_sum = num

num\_of\_sub += 1

return num\_of\_sub <= m

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = Solution()

n =[7,2,5,10,8]

m =2

print("输入数组：",n)

print("输入分组：",m)

print("输出结果：",s.splitArray(n,m))

例393 停车场

3.代码实现

#参数n是层数

#参数num\_rows是每层停车的车位行数

#参数spots\_per\_row是每行停车位的车位数

#参数vehicle是需要停的车辆对象

#返回值是个布尔值，停车成功返回True，否则，False

class VehicleSize:

Motorcycle = 1

Compact = 2

Large = 3

Other = 99

class Vehicle:

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

self.parking\_spots = []

self.spots\_needed = 0

self.size = None

self.license\_plate = None

def get\_spots\_needed(self):

return self.spots\_needed

def get\_size(self):

return self.size

def park\_in\_spot(self, spot):

self.parking\_spots.append(spot)

def clear\_spots(self):

for spot in self.parking\_spots:

spot.remove\_vehicle()

self.park\_sports = []

def can\_fit\_in\_spot(self, spot):

raise NotImplementedError('This method should have implemented.')

class Motorcycle(Vehicle):

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

Vehicle.\_\_init\_\_(self)

self.spots\_needed = 1

self.size = VehicleSize.Motorcycle

def can\_fit\_in\_spot(self, spot):

return True

class Car(Vehicle):

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

Vehicle.\_\_init\_\_(self)

self.spots\_needed = 1

self.size = VehicleSize.Compact

def can\_fit\_in\_spot(self, spot):

return spot.get\_size() == VehicleSize.Large or \

spot.get\_size() == VehicleSize.Compact

class Bus(Vehicle):

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

Vehicle.\_\_init\_\_(self)

self.spots\_needed = 5

self.size = VehicleSize.Large

def can\_fit\_in\_spot(self, spot):

return spot.get\_size() == VehicleSize.Large

class ParkingSpot:

def \_\_init\_\_(self, lvl, r, n, sz):

self.level = lvl

self.row = r

self.spot\_number = n

self.spot\_size = sz

self.vehicle = None

def is\_available(self):

return self.vehicle == None

def can\_fit\_vehicle(self, vehicle):

return self.is\_available() and vehicle.can\_fit\_in\_spot(self)

def park(self, v):

if not self.can\_fit\_vehicle(v):

return False

self.vehicle = v

self.vehicle.park\_in\_spot(self)

return True

def remove\_vehicle(self):

self.level.spot\_freed()

self.vehicle = None

def get\_row(self):

return self.row

def get\_spot\_number(self):

return self.spot\_number

def get\_size(self):

return self.spot\_size

class Level:

def \_\_init\_\_(self, flr, num\_rows, spots\_per\_row):

self.floor = flr

self.spots\_per\_row = spots\_per\_row

self.number\_spots = 0

self.available\_spots = 0;

self.spots = []

for row in range(num\_rows):

for spot in range(0, spots\_per\_row // 4):

sz = VehicleSize.Motorcycle

self.spots.append(ParkingSpot(self, row, self.number\_spots, sz))

self.number\_spots += 1

for spot in range(spots\_per\_row // 4, spots\_per\_row // 4 \* 3):

sz = VehicleSize.Compact

self.spots.append(ParkingSpot(self, row, self.number\_spots, sz))

self.number\_spots += 1

for spot in range(spots\_per\_row // 4 \* 3, spots\_per\_row):

sz = VehicleSize.Large

self.spots.append(ParkingSpot(self, row, self.number\_spots, sz))

self.number\_spots += 1

self.available\_spots = self.number\_spots

def park\_vehicle(self, vehicle):

if self.get\_available\_spots() < vehicle.get\_spots\_needed():

return False

spot\_num = self.find\_available\_spots(vehicle)

if spot\_num < 0:

return False

return self.park\_starting\_at\_spot(spot\_num, vehicle)

def find\_available\_spots(self, vehicle):

spots\_needed = vehicle.get\_spots\_needed()

last\_row = -1

spots\_found = 0

for i in range(len(self.spots)):

spot = self.spots[i]

if last\_row != spot.get\_row():

spots\_found = 0

last\_row = spot.get\_row()

if spot.can\_fit\_vehicle(vehicle):

spots\_found += 1

else:

spots\_found = 0

if spots\_found == spots\_needed:

return i - (spots\_needed - 1)

return -1

def park\_starting\_at\_spot(self, spot\_num, vehicle):

vehicle.clear\_spots()

success = True

for i in range(spot\_num, spot\_num + vehicle.get\_spots\_needed()):

success = success and self.spots[i].park(vehicle)

self.available\_spots -= vehicle.get\_spots\_needed()

return success

def spot\_freed(self):

self.available\_spots += 1

def get\_available\_spots(self):

return self.available\_spots

class ParkingLot:

def \_\_init\_\_(self, n, num\_rows, spots\_per\_row):

self.levels = []

for i in range(n):

self.levels.append(Level(i, num\_rows, spots\_per\_row))

#在停车位上停车，不成功返回False

def park\_vehicle(self, vehicle):

for level in self.levels:

if level.park\_vehicle(vehicle):

return True

return False

#车辆开出停车位

def unpark\_vehicle(self, vehicle):

vehicle.clear\_spots()

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

level=1

num\_rows=1

spots\_per\_row=11

pl=ParkingLot(level,num\_rows,spots\_per\_row)

Car\_1=Car()

Car\_2=Car()

Car\_3=Car()

Car\_4=Car()

Car\_5=Car()

Bus\_1=Bus()

print('停第1辆小汽车：',pl.park\_vehicle(Car\_1))

print('停第2辆小汽车：',pl.park\_vehicle(Car\_2))

print('停第3辆小汽车：',pl.park\_vehicle(Car\_3))

print('停第4辆小汽车：',pl.park\_vehicle(Car\_4))

print('停第5辆小汽车：',pl.park\_vehicle(Car\_5))

print('停第1辆公交车：',pl.park\_vehicle(Bus\_1))

pl.unpark\_vehicle(Car\_5)

print('开出一辆小汽车，停公交车：',pl.park\_vehicle(Bus\_1))

例394 摆动排序

3.代码实现

class Solution(object):

#参数nums是个整数列表

#返回一种排列方案

def wiggleSort( self, nums):

temp = list( sorted(nums))

if len(temp) <= 2:

return temp

count = 1

for i in range(int((len(temp)+1)/2),len(temp)):

temp[2\*count-1],temp[i] = temp[i],temp[2\*count-1]

count = count + 1

return temp

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generator=[1,5,1,1,6,4]

solution= Solution()

print("输入:",generator)

print("输出:",solution.wiggleSort(generator))

例395 实现Trie(前缀树)

3.代码实现

#参数word是输入的一个字符串

#参数prefix是需要查询word的前缀

#返回值是个布尔值，表示功能是否执行成功

class TrieNode:

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

self.children = {}

self.is\_word = False

class Trie:

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

self.root = TrieNode()

def insert(self, word):

node = self.root

for c in word:

if c not in node.children:

node.children[c] = TrieNode()

node = node.children[c]

node.is\_word = True

def find(self, word):

node = self.root

for c in word:

node = node.children.get(c)

if node is None:

return None

return node

def search(self, word):

node = self.find(word)

return node is not None and node.is\_word

def startsWith(self, prefix):

return self.find(prefix) is not None

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Trie()

print('输入: insert("progcode")')

print('输入: search("prog")')

print('输入: startsWith("prog")')

print("输出:", solution.insert("progcode"))

print("输出:", solution.search("prog"))

print("输出:", solution.startsWith("prog"))

例396 Geo哈希I

3.代码实现

#参数latitude，longitude是一个坐标对

#参数precision是一个整数从1~12

#返回值是一个字符串，坐标对对应的字符串

class GeoHash:

def encode(self, latitude, longitude, precision):

\_base32 = "0123456789bcdefghjkmnpqrstuvwxyz"

lat\_bin = self.get\_bin(latitude, -90, 90)

lng\_bin = self.get\_bin(longitude, -180, 180)

hash\_code, b = '', ''

for i in range(30):

b += lng\_bin[i] + lat\_bin[i]

for i in range(0, 60, 5):

hash\_code += \_base32[int(b[i:i + 5], 2)]

return hash\_code[:precision]

def get\_bin(self, value, left, right):

b = ''

for i in range(30):

mid = (left + right) / 2.0

if value > mid:

left = mid

b += '1'

else:

right = mid

b += '0'

return b

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution= GeoHash()

lat = -90

lng = 180

precision = 12

print("输入 :lat=",lat,"lng=",lng,"precision =",precision)

print("输出 :",solution.encode(lat,lng,precision))

例397 Geo哈希II

3.代码实现

#参数geohash是一个32位字符串的哈希坐标

#返回值location是对应的坐标

class GeoHash:

def decode(self, geohash):

\_base32 = "0123456789bcdefghjkmnpqrstuvwxyz"

b = ""

for c in geohash:

b += self.i2b(\_base32.find(c))

odd = ''.join([b[i] for i in range(0, len(b), 2)])

even = ''.join([b[i] for i in range(1, len(b), 2)])

location = []

location.append(self.get\_location(-90.0, 90.0, even))

location.append(self.get\_location(-180.0, 180.0, odd))

return location

def i2b(self, val):

b = ""

for i in range(5):

if val % 2:

b = '1' + b

else:

b = '0' + b

val //= 2

return b

def get\_location(self, start, end, string):

for c in string:

mid = (start + end) / 2.0

if c == '1':

start = mid

else:

end = mid

return (start + end) / 2.0

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution= GeoHash()

generator="wx4g0s"

print("输入 :",generator)

print("输出 :",solution.decode(generator))

例398 友谊服务

3.代码实现

#参数user\_id是个整数，用户ID

#返回值results是个整数数组，由该用户所有的followers组成

class FriendshipService:

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

self.followers = dict()

self.followings = dict()

def getFollowers(self, user\_id):

if user\_id not in self.followers:

return []

results = list(self.followers[user\_id])

results.sort()

return results

def getFollowings(self, user\_id):

if user\_id not in self.followings:

return []

results = list(self.followings[user\_id])

results.sort()

return results

def follow(self, to\_user\_id, from\_user\_id):

if to\_user\_id not in self.followers:

self.followers[to\_user\_id] = set()

self.followers[to\_user\_id].add(from\_user\_id)

if from\_user\_id not in self.followings:

self.followings[from\_user\_id] = set()

self.followings[from\_user\_id].add(to\_user\_id)

def unfollow(self, to\_user\_id, from\_user\_id):

if to\_user\_id in self.followers:

if from\_user\_id in self.followers[to\_user\_id]:

self.followers[to\_user\_id].remove(from\_user\_id)

if from\_user\_id in self.followings:

if to\_user\_id in self.followings[from\_user\_id]:

self.followings[from\_user\_id].remove(to\_user\_id)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = FriendshipService()

solution.follow(1,3)

solution.getFollowers(1)

print('输入:\nsolution.follow(1, 3)\n'

'solution.getFollowers(1)\nsolution.getFollowings(3)')

print('solution.follow(2, 3)\nsolution.getFollowing(3)')

print('solution.unfollow(1, 3)\nsolution.getFollowings(3)')

print("输出:")

print('3跟随1后，1的跟随者:',solution.getFollowers(1))

solution.getFollowings(3)

print('3跟随1后，3所跟随的:',solution.getFollowings(3))

solution.follow(2, 3)

solution.getFollowings(3)

print('3跟随2后, 3所跟随的:',solution.getFollowings(3))

solution.unfollow(1, 3)

solution.getFollowings(3)

print('3解除跟随1后, 3所跟随的:',solution.getFollowings(3))

例399 DNA重复问题

3.代码实现

#参数s是一个字符串，代表DNA序列  
#返回所有10个字母长的序列  
class Solution:  
 def findRepeatedDna(self, s):  
 dict = {}  
 for i in range(len(s) - 9):  
 key = s[i:i + 10]  
 if key not in dict:  
 dict[key] = 1  
 else:  
 dict[key] += 1  
 result = []  
 for element in dict:  
 if dict[element] > 1:  
 result.append(element)  
 return result  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 s = "AAAAACCCCCAAAAACCCCCCAAAAAGGGTTT"  
 #创建对象  
 solution = Solution()  
 print("输入字符串:", s)  
 print("输出结果:", solution.findRepeatedDna(s))

例400 字模式

3.代码实现

#参数pattern是一个字符串，代表了给定模式的字符串  
#参数teststr是一个字符串，代表了匹配的字符串  
#返回值是一个布尔值，代表了给定模式的字符串和匹配的字符串是否匹配  
class Solution:  
 def wordPattern(self, pattern, teststr):  
 map = {}  
 myset = set() #set用来预防ab=“cat cat”这种情况  
 teststr = teststr.split(' ')  
 if len(pattern) != len(teststr): #如果长度不等直接返回False  
 return False  
 for i in range(len(pattern)):  
 if pattern[i] not in map:  
 if teststr[i] not in myset:

#如果set中没有就代表此时的pattern和teststr都是新的，添加  
 map[pattern[i]] = teststr[i]  
 myset.add(teststr[i])  
 else: #如果set中存在，代表之前有的pattern已经表示了teststr，返回False  
 return False  
 if teststr[i] != map[pattern[i]]:  
 return False  
 return True  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":

pattern = "abba"

str = "dog cat cat dog"

#创建对象

solution = Solution()

print("输入模式",pattern,",字符串str=", str)

print("输出结果:",solution.wordPattern(pattern, str))

例401 字符同构

3.代码实现

#参数s是一个字符串  
#参数t是一个字符串  
#返回一个布尔值，如果s中的字符可以被取代就返回True  
class Solution:  
 def isUniqueMapping(self, s, t):  
 if len(s) != len(t):  
 return False  
 charMap = dict()  
 for i in range(len(s)):  
 if s[i] not in charMap:  
 charMap[s[i]] = t[i]  
 else:  
 if t[i] != charMap[s[i]]:  
 return False  
 return True  
 def isIsomorphic(self, s, t):  
 return self.isUniqueMapping(s, t) and self.isUniqueMapping(t, s)  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 s = "paper"  
 t = "title"  
 #创建对象  
 solution = Solution()  
 print("初始两个字符串：s=:", s, ",t=:", t)  
 print("输出结果：", solution.isIsomorphic(s, t))

例402 课程表

3.代码实现

#采用utf-8编码格式

#参数courses是课程持续时间和结束时间

#返回值是可以上的最大数量课程数

import heapq

class Solution:

def scheduleCourse(self, courses):

if courses == None or len(courses) == 0:

return 0

courses.sort(key = lambda x : x[1])

queue = []

time = 0

for i in range(len(courses)):

if time + courses[i][0] <= courses[i][1]:

time += courses[i][0]

heapq.heappush(queue, -courses[i][0])

elif queue and courses[i][0] < (-queue[0]):

time += courses[i][0] - (-queue[0])

heapq.heapreplace(queue, -courses[i][0])

return len(queue)

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

List1 = [[300, 100],[100, 100],[500, 800]]

List2 = [[100, 200],[200, 1300],[1000, 1250],[2000, 3200]]

print("输入："+str(List1))

print("输出："+str(temp.scheduleCourse(List1)))

print("输入："+str(List2))

print("输出："+str(temp.scheduleCourse(List2)))

例403 小行星的碰撞

3.代码实现

#参数为一个整数列表  
#返回一个整数列表  
class Solution:  
 def asteroidCollision(self, asteroids):  
 stack = []  
 result = []  
 for asteroid in asteroids:  
 if asteroid > 0:  
 stack.append(asteroid)  
 else:  
 while stack and stack[-1] < abs(asteroid):  
 stack.pop()  
 if not stack:  
 result.append(asteroid)  
 else:  
 if stack[-1] == abs(asteroid):  
 stack.pop()  
 continue  
 else:  
 continue  
 return result + stack  
#主函数  
if \_\_name\_\_=="\_\_main\_\_":  
 asteroids=[5,10,-5]  
 #创建对象  
 solution=Solution()  
 print("输入顺序：",asteroids)  
 print("运行结果：",solution.asteroidCollision(asteroids))

例404 K个空的位置

3.代码实现

#参数flowers是哪一天将要开放的位置  
#参数k是一个整数  
#返回值是刚好有两朵花开放的那一天  
import queue

class Solution:  
 def kEmptySlots(self, flowers, k):  
 if flowers is None or len(flowers) <= 1:  
 return -1  
 q = queue.Queue()  
 min\_val, max\_val = min(flowers[0], flowers[1]), max(flowers[0], flowers[1])  
 q.put((min\_val, max\_val))  
 n = len(flowers)  
 day = 2  
 for i in range(2, n):  
 new\_flower = flowers[i]  
 for j in range(q.qsize()):  
 left, right = q.get()  
 if right - left == k + 1:  
 return day  
 if new\_flower > left and new\_flower < right:  
 q.put((left, new\_flower))  
 q.put((new\_flower, right))  
 elif right - left > k + 1:  
 q.put((left, right))  
 if new\_flower < min\_val:  
 q.put((new\_flower, min\_val))  
 min\_val = new\_flower  
 if new\_flower > max\_val:  
 q.put((max\_val, new\_flower))  
 max\_val = new\_flower  
 day += 1  
 while not q.empty():  
 left, right = q.get()  
 if left - right == k:  
 return day  
 return -1  
#主函数  
if \_\_name\_\_=="\_\_main\_\_":  
 flowers=[1,3,2]  
 k=1  
 #创建对象  
 solution=Solution()  
 print("输入：flowers=",flowers,"k=",k )  
 print("输出：恰好有两朵花处于盛开的状态是第%d天"% solution.kEmptySlots(flowers,k))

例405 逆序对

3.代码实现

#参数为一个数组  
#返回的是逆序对的总数

class Solution:  
 def reversePairs(self, A):  
 self.tmp = [0] \* len(A)  
 return self.mergeSort(A, 0, len(A) - 1)  
 def mergeSort(self, A, l, r):  
 if l >= r:  
 return 0  
 m = (l + r) >> 1  
 ans = self.mergeSort(A, l, m) + self.mergeSort(A, m + 1, r)  
 i, j, k = l, m + 1, l  
 while i <= m and j <= r:  
 if A[i] > A[j]:  
 self.tmp[k] = A[j]  
 j += 1  
 ans += m - i + 1  
 else:  
 self.tmp[k] = A[i]  
 i += 1  
 k += 1  
 while i <= m:  
 self.tmp[k] = A[i]  
 k += 1  
 i += 1  
 while j <= r:  
 self.tmp[k] = A[j]  
 k += 1  
 j += 1  
 for i in range(l, r + 1):  
 A[i] = self.tmp[i]  
 return ans  
#主函数  
if \_\_name\_\_=="\_\_main\_\_":  
 arr=[2, 4, 1, 3, 5]  
 #创建对象  
 solution=Solution()  
 print("输入数组顺序：",arr)  
 print("逆序对总数:",solution.reversePairs(arr))

例406 任务调度器

3.代码实现

#参数tasks是一个给定的字符数组，表示CPU需要执行的任务

#参数n是一个非负冷却间隔

#返回值是CPU完成所有给定任务所需的最小间隔时间

from collections import Counter

class Solution:

def leastInterval(self, tasks, n):

ct = Counter(tasks)

max\_ct = max(ct.values())

top\_freq\_task = [t for t in ct if ct[t] == max\_ct]

#步骤1：计数最大频率任务的循环次数为max\_ct

#步骤2：计数间隔总和为(max\_ct - 1) \* n

#步骤3：如果有绑定最高频率的任务，那么在步骤1的最后一个循环后，为每个

#额外的绑定最高频率的任务添加一个尾循环

return max\_ct + (max\_ct - 1) \* n + len(top\_freq\_task) - 1

#主函数

if \_\_name\_\_ == "\_\_main\_\_":

tasks = ['A', 'A', 'A', 'B', 'B', 'B']

n = 2

#创建对象

solution = Solution()

print("初始任务：", tasks, ",给定n=", n)

print("最小间隔数：", solution.leastInterval(tasks, n))

例407 下一个排列

3.代码实现

#参数nums是一个整数数组  
#返回值下一个排列  
class Solution:  
 def nextPermutation(self, nums):  
 #倒序遍历  
 for i in range(len(nums)-1, -1, -1):  
#找到第一个数值变小的点，这样代表右边有大的可以和它交换，而且可以保证是下一个排列  
 if i > 0 and nums[i] > nums[i-1]:  
#找到后再次倒序遍历，找到第一个比刚才那个数值大的点，互相交换  
 for j in range(len(nums)-1, i-1, -1):  
 if nums[j] > nums[i-1]:  
 nums[j], nums[i-1] = nums[i-1], nums[j]  
#因为之前保证了，右边这段数从右到左是一直变大的，所以直接双指针反转  
 left, right = i, len(nums)-1  
 while left <= right:  
 nums[left], nums[right] = nums[right], nums[left]  
 left += 1  
 right -= 1  
 return nums  
 #如果循环结束了，没找到能替换的数，表示序列已经是最大的

nums.reverse()  
 return nums  
#主函数  
if \_\_name\_\_=="\_\_main\_\_":  
 nums = [1,2,3]  
 #创建对象  
 solution=Solution()  
 print("输入数组：",nums)  
 print("下一个排列：",solution.nextPermutation(nums))

例408 范围加法

3.代码实现

#参数length是数组的长度  
#参数updates是更新操作  
#返回执行k个更新操作后的新数组  
class Solution:

def getModifiedArray(self, length, updates):  
 result = [0 for i in range(length)]  
 operation = result + [0]  
 for start, end, val in updates:  
 operation[start] += val  
 operation[end + 1] -= val  
 for index in range(len(result)):  
 if index == 0:  
 result[index] = operation[index]  
 continue  
 result[index] = operation[index] + result[index - 1]  
 return result  
#主函数  
if \_\_name\_\_=="\_\_main\_\_":  
 length = 5  
 updates=[[1, 3, 2],[2, 4, 3],[0, 2, -2]]  
 #创建对象  
 solution=Solution()  
 print("输入长度：",length,",更新数组：",updates)  
 print("输出结果：",solution.getModifiedArray(length,updates))

例409 n皇后问题

3.代码实现

#参数n是皇后的数量

#返回值是所有不同的解决方法

class Solution:

def solveNQueens(self, n):

results = []

self.search(n, [], results)

return results

def search(self, n, cols, results):

row = len(cols)

if row == n:

results.append(self.draw\_chessboard(cols))

return

for col in range(n):

if not self.is\_valid(cols, row, col):

continue

cols.append(col)

self.search(n, cols, results)

cols.pop()

def draw\_chessboard(self, cols):

n = len(cols)

board = []

for i in range(n):

row = ['Q' if j == cols[i] else '.' for j in range(n)]

board.append(''.join(row))

return board

def is\_valid(self, cols, row, col):

for r, c in enumerate(cols):

if c == col:

return False

if r - c == row - col or r + c == row + col:

return False

return True

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

n = int(input("请输入一个正整数："))

solution = Solution()

print(n, "皇后问题的解：", solution.solveNQueens(n))

例410 用递归打印数字

3.代码实现

#参数n是一个整数

#返回值是一个整数数组，储存了从1到最大的n位整数

class Solution:

def numbersByRecursion(self, n):

top = pow(10, n)

rt = []

for i in range(1, top):

rt.append(i)

return rt

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

n = int(input("请输入一个正整数:"))

solution = Solution()

print("从1到最大的", n, "位数：", solution.numbersByRecursion(n))

例411 推荐朋友

3.代码实现

#参数friends是朋友列表  
#参数user是使用者的ID  
#返回最可能认识的人  
class Solution:  
 def recommendFriends(self, friends, user):  
 n = len(friends)  
 userSet = {}  
 ans = 0  
 idx = -1  
 for i in friends[user]:  
 userSet[i] = i  
 for i in range(n):  
 if i == user or i in userSet:  
 continue  
 t = 0  
 for j in friends[i]:  
 if j in userSet:  
 t = t + 1  
 if t > ans:  
 ans = t  
 idx = i  
 return idx  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":

friends = [[1, 2, 3], [0, 4], [0, 4], [0, 4], [1, 2, 3]]

user = 0

#创建对象

solution = Solution()

print("初始朋友列表:", friends, ",给定初始的user:", user)

print("user最可能认识的人：", solution.recommendFriends(friends, user))

例412 Nuts和Bolts的问题

3.代码实现

#参数nuts是一个整数数组

#参数bolts是一个整数数组

#参数compare是一个比较器的实例

#返回比较后的值

class Comparator:

def cmp(self, a, b):

if a > b:

return 1

elif a == b:

return 0

elif a < b:

return -1

else:

return 2

class Solution:

def sortNutsAndBolts(self, nuts, bolts, compare):

self.quick\_sort(nuts, bolts, 0, len(nuts) - 1, compare.cmp)

def quick\_sort(self, nuts, bolts, start, end, cmp):

if start >= end:

return

left, right = start, end

index = self.partition(bolts, left, right, nuts[(left + right) // 2], cmp)

self.partition(nuts, left, right, bolts[index], cmp)

self.quick\_sort(nuts, bolts, start, index - 1, cmp)

self.quick\_sort(nuts, bolts, index + 1, end, cmp)

def partition(self, arr, start, end, pivot, cmp):

left, right = start, end

for i in range(left, right + 1):

if cmp(arr[i], pivot) == 0 or cmp(pivot, arr[i]) == 0:

arr[i], arr[left] = arr[left], arr[i]

left += 1

break

while left <= right:

while left <= right and (cmp(arr[left], pivot) == -1 or cmp(pivot, arr[left]) == 1):

left += 1

while left <= right and (cmp(arr[right], pivot) == 1 or cmp(pivot, arr[right]) == -1):

right -= 1

if left <= right:

arr[left], arr[right] = arr[right], arr[left]

left, right = left + 1, right - 1

arr[start], arr[right] = arr[right], arr[start]

return right

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

nuts = ['ab', 'bc', 'dd', 'gg']

bolts = ['AB', 'GG', 'DD', 'BC']

compaer = Comparator()

print("初始数组：")

print("nuts={}".format(nuts))

print("bolts={}".format(bolts))

solution = Solution()

solution.sortNutsAndBolts(nuts, bolts, compaer)

print("结果：")

print("nuts={}".format(nuts))

print("bolts={}".format(bolts))

例413 Fizz Buzz问题

3.代码实现

#采用utf-8编码格式

#参数n是描述中的一个整数

#返回一个字符列表

#如果n=7，代码应该返回["1", "2", "fizz", "4", "buzz", "fizz", "7"]

class Solution:

def fizzBuzz(self, n):

results = []

for i in range(1, n+1):

if i % 15 == 0:

results.append("fizz buzz")

elif i % 5 == 0:

results.append("buzz")

elif i % 3 == 0:

results.append("fizz")

else:

results.append(str(i))

return results

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

nums1 = 10

nums2 = 13

print ("输入："+str(nums1))

print ("输出："+str(temp.fizzBuzz(nums1)))

print ("输入："+str(nums2))

print ("输出："+str(temp.fizzBuzz(nums2)))

例414 通配符匹配

3.代码实现

#采用utf-8编码格式

#参数s是一个字符串

#参数p是一个包含"?" 和"\*"的字符串

#返回一个布尔值

class Solution:

def isMatch(self, s, p):

n = len(s)

m = len(p)

f = [[False] \* (m + 1) for i in range(n + 1)]

f[0][0] = True

if n == 0 and p.count('\*') == m:

return True

for i in range(0, n + 1):

for j in range(0, m + 1):

if i > 0 and j > 0:

f[i][j] |= f[i-1][j-1] and (s[i-1] == p[j-1] or p[j - 1] in ['?', '\*'])

if i > 0 and j > 0:

f[i][j] |= f[i - 1][j] and p[j - 1] == '\*'

if j > 0:

f[i][j] |= f[i][j - 1] and p[j - 1] == '\*'

return f[n][m]

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

string1 = "bb"

string2 = "b"

print(("输入："+string1+" "+string2))

print(("输出："+str(temp.isMatch(string1,string2))))

例415 最大子矩阵

3.代码实现

#参数matrix是一个给定的矩阵

#返回值是最大可能总和值

class Solution:

def maxSubmatrix(self, matrix):

if matrix is None or len(matrix) == 0:

return 0

m, n = len(matrix), len(matrix[0])

max\_sum = 0

for i in range(n):

for j in range(i, n):

temp\_array = [sum(matrix[k][i:j+1]) for k in range(m)]

max\_sum = max(self.maxSubarray(temp\_array), max\_sum)

return max\_sum

def maxSubarray(self, array):

running\_sum = 0

max\_sum = 0

for a in array:

running\_sum = max(running\_sum + a, a)

max\_sum = max(running\_sum, max\_sum)

max\_sum = max(running\_sum, max\_sum)

return max\_sum

#主函数

if \_\_name\_\_=="\_\_main\_\_":

matrix = [[1, 3, -1], [2, 3, -2], [-1, -2, -3]]

#创建对象

solution=Solution()

print("输入数组：",matrix)

print("子矩阵组成最大可能的和:",solution.maxSubmatrix(matrix))

例416 更新二进制位

3.代码实现

#参数N、M是两个整数

#参数i、j代表两个二进制位的位置

#返回一个整数

class Solution:

def updateBits(self, n, m, i, j):

a = list()

for k in range(32):

a.append(n % 2)

n //= 2

for k in range(i, j + 1):

a[k] = m % 2

m //= 2

n = 0

for k in range(31):

if a[k] == 1:

n |= (1 << k)

if a[31] == 1:

n -= 1 << 31

return n

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

n = 1024; m = 21

i = 2; j = 6

print(("输入：n="+str(n)+", m="+str(m)+", i="+str(i)+", j="+str(j)))

print(("输出："+str(temp.updateBits(n,m,i,j))))

例417 两个数组最小差问题

3.代码实现

#参数A、B是两个整数数组  
#返回一个整数  
class Solution:  
 def smallestDifference(self, A, B):  
 C = []  
 for x in A:  
 C.append((x, 'A'))  
 for x in B:  
 C.append((x, 'B'))  
 C.sort()  
 diff = 0x7fffffff  
 cnt = len(C)  
 for i in range(cnt - 1):  
 if C[i][1] != C[i + 1][1]:  
 diff = min(diff, C[i + 1][0] - C[i][0])  
 return diff  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 A=[3,4,6,7]  
 B=[2,3,8,9]  
 #创建对象  
 solution = Solution()  
 print("输入数组分别是：",A,B )  
 print("两个数组之间最小差:", solution.smallestDifference(A,B))

例418 单词反转

3.代码实现

#采用utf-8编码格式

#参数A是一个字符串

#返回一个字符串

class Solution:

def reverseWords(self, s):

return ' '.join(reversed(s.strip().split()))

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

string1 = "hello world"

string2 = "python learning"

print(("输入："+string1))

print(("输出："+temp.reverseWords(string1)))

print(("输入："+string2))

print(("输出："+temp.reverseWords(string2)))

例419 四的乘方

3.代码实现

class Solution:

#参数num为整数

#返回布尔类型

def isPowerOfFour(self, num):

basic = 4

i = 0

while basic\*\*i <= num:

if basic\*\*i == num:

return True

i += 1

return False

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

solution = Solution()

Test\_in = 16

print("输入：",Test\_in)

print("输出：",solution.isPowerOfFour(Test\_in))

例420 A + B 问题

3.代码实现

class Solution:

#参数a为整数

#参数b为整数

#返回整数

def aplusb(self, a, b):

return a + b

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

a = 1

b = 2

print('输入:a =',a,'b=',b)

print('输出:a+b=',temp.aplusb(a,b))

例421 尾部的零

3.代码实现

class Solution:

#参数n为整数

#返回整数

def trailingZeros(self, n: int) -> int:

x=n//5

count=0

while x >0:

count =x+count

x=x//5

return count

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

a = 100

print('输入:a =',a)

print('输出:',temp.trailingZeros(a))

例422 移动的圆

3.代码实现

import math

class Solution:

#参数position为圆A,B和点P的值

#返回值为1或者-1

#叉积AB×AC

def xmult(self, B, C, A):

return (B[0] - A[0])\*(C[1] - A[1]) - (C[0] - A[0])\*(B[1] - A[1])

#两点间距离

def distance(self, A, B):

return math.sqrt((A[0] - B[0])\*(A[0] - B[0]) + (A[1] - B[1])\*(A[1] - B[1]))

#点A到直线BC距离

def dis\_ptoline(self, A, B, C):

return abs(self.xmult(A,B,C))/self.distance(B,C)

def IfIntersect(self, position):

A = [position[0], position[1]]

ra = position[2]

B = [position[3], position[4]]

rb = position[5]

P = [position[6], position[7]]

#过点B作直线AP垂线，M为该垂线上一点（A和P不重合时M点不与B重合）

M = [B[0] - (P[1] - A[1]), B[1] + (P[0] - A[0])]

dmin = 0.0

dmax = 0.0

#若圆A移动过程中会经过B点到直线AP垂线的交点

if self.xmult(A, B, M) \* self.xmult(B, P, M) > 0 :

dmin = self.dis\_ptoline(B, A, P)

else :

dmin = min(self.distance(A, B), self.distance(P, B))

dmax = max(self.distance(A, B), self.distance(P, B))

if dmin > ra + rb or dmax < abs(ra - rb):

return -1

return 1

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

a =[0,0,2.5,3,2,0.5,0,2]

print('输入:a =',a)

print('输出:',temp.IfIntersect(a))

例423 列表扁平化

3.代码实现

class Solution(object):

#参数nestedList为列表

#返回整数列表

def flatten(self, nestedList):

stack = [nestedList]

flatten\_list = []

while stack:

top = stack.pop()

if isinstance(top, list):

for elem in reversed(top):

stack.append(elem)

else:

flatten\_list.append(top)

return flatten\_list

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

a =[[1,1],2,[1,1]]

print('输入:a =',a)

print('输出:',temp.flatten(a))

例424 判断数字与字母字符

3.代码实现

class Solution:

#参数c为字符型

#返回布尔类型

''' if c.isalpha() or c.isdigit():

或者

if c.isalnum():

注意字符编码应为utf-8否则某些特殊情况下函数失效'''

def isAlphanumeric(self, c):

if c.isalnum():

return True

else:

return False

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

c = '1'

print('输入:c =',c)

print('输出:',temp.isAlphanumeric(c))

例425 打印X

3.代码实现

class Solution:

#参数n为整数

#返回字符串列表

def printX(self, n):

A = []

for i in range(n):

lin\_n = ""

for j in range(n):

if j==i or j==n-i-1:

lin\_n = lin\_n + "X"

else:

lin\_n = lin\_n + " "

A.insert(i+j,lin\_n)

return A

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

n = 3

print('输入: n =',n)

print('输出:',temp.printX(n))

例426 内积

3.代码实现

class Solution:

#参数 A为数组

#参数B为数组

#返回整数

def getMaxInnerProduct(self, A, B):

# A长度

n = len(A)

# B长度

K = len(B)

#初始化dp数组

# dp[i][j]表示从左边取i个数，从右边取j个数的最大内积

dp = [[0] \* (K + 1) for i in range(K + 1)]

#枚举dp[i][j]

for i in range(K + 1):

for j in range(K + 1):

#从左边和右边取数总数不超过K个

if i + j > K or i + j > n:

continue

# dp数组边界条件，从左右都不取数时，dp[0][0]=0

if i == 0 and j == 0:

dp[i][j] = 0

continue

#从左边取的i 更新dp[i][j]

if i != 0:

dp[i][j] = max(dp[i][j], dp[i - 1][j] + A[i - 1] \* B[i + j - 1])

#从右边取的j，更新dp[i][j]

if j != 0:

dp[i][j] = max(dp[i][j], dp[i][j - 1] + A[n - j] \* B[i + j - 1])

#枚举从左边取了多少，找最大的内积

ans = 0

for i in range(K + 1):

ans = max(ans, dp[i][K - i])

return ans

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

A=[2,3,5,1]

B=[2,1]

print('输入:',A,B)

print('输出:',temp.getMaxInnerProduct(A,B))

例427 abc串

3.代码实现

class Solution:

#参数n为整数

#参数k为整数

#返回字符串

def kthString(self, n, k):

#判断 k 是否超出不同字符串的个数

#长为 n 的字符串长度应等于 3 \* (2 ^ (n - 1))

# n 控制在 62 以内是因为计算 2 的幂可能会溢出和时间超限

if n <= 62 and 3 \* (2 \*\* (n - 1)) < k:

return ""

result = ""

#计算第一个字符

if n >= 62:

result += 'a'

elif k <= 2 \*\* (n - 1):

result += 'a'

elif k <= 2 \* (2 \*\* (n - 1)):

result += 'b'

k -= 2 \*\* (n - 1)

else:

result += 'c'

k -= 2 \* (2 \*\* (n - 1))

#计算后续字符

for i in range(1, n):

# position = 0代表这个位置填较小的字符，1填较大的

position = 0

exponent = n - i

if exponent < 62 and k > 2 \*\* (exponent - 1):

position = 1

k -= 2 \*\* (exponent - 1)

temp = "abc"

temp = temp.replace(result[i - 1], '', 1)

result += temp[position]

return result

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

n=3

k=6

print('输入:',n,k)

print('输出:',temp.kthString(n,k))

例428 最大子数组 I

3.代码实现

class Solution:

#参数nums为整数数组

#返回整数

def maxTwoSubArrays(self, nums):

n = len(nums)

a = nums[:]

aa = nums[:]

for i in range(1, n):

a[i] = max(nums[i], a[i-1] + nums[i])

aa[i] = max(a[i], aa[i-1])

b = nums[:]

bb = nums[:]

for i in range(n-2, -1, -1):

b[i] = max(b[i+1] + nums[i], nums[i])

bb[i] = max(b[i], bb[i+1])

mx = -65535

for i in range(n - 1):

mx = max(aa[i]+b[i+1], mx)

return mx

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

nums=[1,3,-1,2,-1,2]

print('输入:',nums)

print('输出:',temp.maxTwoSubArrays(nums))

例429 最大子数组 II

3.代码实现

class Solution:

#参数nums为整数数组

#参数k为整数

#返回整数

def maxKSubArrays(self, nums, K):

dp = [[[-float('inf')] \* 2 for \_ in range(K + 1)] for \_\_ in range(len(nums) + 1)]

dp[0][0][0] = 0

dp[0][0][1] = 0

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

dp[i][0][0] = 0

for j in range(1, K + 1):

dp[i][j][0] = max(dp[i - 1][j][0], dp[i - 1][j][1])

dp[i][j][1] = max(dp[i - 1][j - 1][0] + nums[i - 1], dp[i - 1][j - 1][1] + nums[i - 1], dp[i - 1][j][1] + nums[i - 1])

return max(dp[len(nums)][K][0], dp[len(nums)][K][1])

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

nums=[1,2,3]

k=1

print('输入:nmms=',nums,',k=',k)

print('输出:',temp.maxKSubArrays(nums,k))

例430 最小子数组

3.代码实现

class Solution:

#参数nums是整数数组

#返回整数

def minSubArray(self, nums):

sum = 0

minSum = nums[0]

maxSum = 0

for num in nums:

sum += num

if sum - maxSum < minSum:

minSum = sum - maxSum

if sum > maxSum:

maxSum = sum

return minSum

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

nums=[1,-1,-2,1]

print('输入:nmms=',nums)

print('输出:',temp. minSubArray(nums))

例431 最大子数组差

3.代码实现

class Solution:

#参数nums为整数数组

#返回整数

def maxDiffSubArrays(self, nums):

n = len(nums)

mx1 = [0]\*n

mx1[0] = nums[0]

mn1 = [0]\*n

mn1[0] = nums[0]

forward = [mn1[0], mx1[0]]

array\_f = [0]\*n

array\_f[0] = forward[:]

for i in range(1, n):

mx1[i] = max(mx1[i-1] + nums[i], nums[i])

mn1[i] = min(mn1[i-1] + nums[i], nums[i])

forward = [min(mn1[i], forward[0]), max(mx1[i], forward[1])]

array\_f[i] = forward[:]

mx2 = [0]\*n

mx2[n-1] = nums[n-1]

mn2 = [0]\*n

mn2[n-1] = nums[n-1]

backward = [mn2[n-1], mx2[n-1]]

array\_b = [0]\*n

array\_b[n-1] = backward[:]

for i in range(n-2, -1, -1):

mx2[i] = max(mx2[i+1] + nums[i], nums[i])

mn2[i] = min(mn2[i+1] + nums[i], nums[i])

backward = [min(mn2[i], backward[0]), max(mx2[i], backward[1])]

array\_b[i] = backward[:]

result = -65535

for i in range(n-1):

result = max(result, abs(array\_f[i][0] - array\_b[i+1][1]), abs(array\_f[i][1] - array\_b[i+1][0]))

return result

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

nums=[1,2,-3,1]

print('输入:nmms=',nums)

print('输出:',temp.maxDiffSubArrays(nums))

例432 k数之和

3.代码实现

class Solution:

#参数A为整数数组

#参数k为整数

#参数target为整数

#返回整数

def kSumII(self, A, k, target):

A = sorted(A)

subsets = []

self.dfs(A, 0, k, target, [], subsets)

return subsets

def dfs(self, A, index, k, target, subset, subsets):

if k == 0 and target == 0:

subsets.append(list(subset))

return

if k == 0 or target <= 0:

return

for i in range(index, len(A)):

subset.append(A[i])

self.dfs(A, i + 1, k - 1, target - A[i], subset, subsets)

subset.pop()

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

A=[1,2,3,4]

k=2

target=5

print('输入:A=',A,',k=',k,',target=',target)

print('输出:',temp.kSumII(A, k, target))

例433 O(1)时间检测2的幂次

3.代码实现

class Solution:

#参数n为整数

#返回True或False

def checkPowerOf2(self, n):

ans = 1

for i in range(31):

if ans == n:

return True

ans = ans << 1

return False

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

n = 4

print('输入: n=',n)

print('输出:',temp.checkPowerOf2(n))

434 最大数

3.代码实现

import functools

class Solution:

#参数nums为整数数组

#返回字符串

#比较函数

def compare(self, a, b):

if a + b > b + a:

return -1

return 1

def largestNumber(self, nums):

string = []

#把整型转换成字符串

for i in nums:

string.append(str(i))

#按最优策略排序

string.sort(key = functools.cmp\_to\_key(self.compare))

ans = ""

for i in string:

ans += i

#除去有多余前导0的情况

if ans[0] == '0':

return "0"

return ans

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

nums = [1,20,23,4,8]

print('输入: nums=',nums)

print('输出:',temp.largestNumber(nums))

435 插入5

3.代码实现

class Solution:

#参数a为整数

#返回整数

def InsertFive(self, a):

string = ''

ans = 0

n = 0

flag = False

if a >= 0:

string = str(a)

n = len(string)

for i in range(n):

if (ord(string[i]) - ord('0')) < 5 and flag == False:

ans = ans \* 10 + 5

flag = True

ans = ans \* 10 + ord(string[i]) - ord('0')

if flag == False:

ans = ans \* 10 + 5

else:

a = -a

string = str(a)

n = len(string)

for i in range(n):

if ord(string[i]) - ord('0') > 5 and flag == False:

ans = ans \* 10 + 5

flag = True

ans = ans \* 10 + ord(string[i]) - ord('0')

if flag == False:

ans = ans \* 10 + 5

ans = -ans

return ans

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

a = 234

print('输入: a=',a)

print('输出:',temp.InsertFive(a))

436 寻找单词

3.代码实现

class Solution:

#参数s为字符串

#参数d为字典

#返回字符串

def is\_subsequence(self, s, t):

i, j = 0, 0

while i < len(s) and j < len(t):

if (s[i] == t[j]):

i += 1

j += 1

else:

i += 1

return j == len(t)

def findWords(self, s, d):

result = []

for word in d:

if self.is\_subsequence(s, word):

result.append(word)

return result

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

s="bcogtadsjofisdhklasdj"

d=["book","code","tag"]

print('输入: s=',s,',d=',d)

print('输出:',temp.findWords(s,d))

437 判断连接

3.代码实现

class Solution:

#参数arr为矩阵

#参数k为整数

#返回布尔类型

def dfs(self, arr, x, y, k):

arr[x][y] = -1;

dx = [0, 0, 1, -1]

dy = [1, -1, 0, 0]

for i in range(0,4):

x1 = x + dx[i]

y1 = y + dy[i]

if x1 >= 0 and y1 >= 0 and x1 < len(arr) and y1 < len(arr[0]) and arr[x1][y1] == k:

self.dfs(arr, x1, y1, k);

def judgeConnection(self, arr, k):

sum = 0;

for i in range(0,len(arr)):

for j in range(0,len(arr[0])):

if arr[i][j] == k:

self.dfs(arr, i, j, k)

sum += 1

if sum >= 2:

return False

return True

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

arr=[[2,2,2,0],[0,0,0,2],[0,1,0,2],[1,1,1,2]]

k=1

print('输入: arr=',arr,',k=',k)

print('输出:',temp.judgeConnection(arr,k))

例438 冰雹猜想

3.代码实现

class Solution:

#参数num为整数

#返回整数

def getAnswer(self, num):

count = 0

while num != 1:

if num % 2 == 1:

num = num \* 3 + 1

else:

num /= 2

count += 1

return count

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

num=4

print('输入: num=',num)

print('输出:',temp.getAnswer(num))

例439 链表求和

3.代码实现

class ListNode(object):

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

self.val = val

self.next = next

class Solution:

#参数l1为第一个链表

#参数l2为第二个链表

#返回列表

#反转链表

def reverse(self, l):

# pre->cur反转为cur→pre,next用于遍历原链表

pre = None

cur = l

next = cur.next

while next:

cur.next = pre

pre = cur

cur = next

next = next.next

cur.next = pre

return cur

def addLists2(self, l1, l2):

l1 = self.reverse(l1)

l2 = self.reverse(l2)

ans = ListNode(0)

cur = ans

# pre用于最后删去最高位为0的节点

pre = None

# l1和l2逐位从低位到高位相加，直到l1或l2到最高位

while l1 and l2:

# sum = 进位 + 二者之和

sum = cur.val + l1.val + l2.val

cur.val = sum % 10

cur.next = ListNode(sum // 10)

l1 = l1.next

l2 = l2.next

pre = cur

cur = cur.next

#如果l1 或 l2还有更高位，继续加到答案链表

while l1:

sum = cur.val + l1.val

cur.val = sum % 10

cur.next = ListNode(sum // 10)

l1 = l1.next

pre = cur

cur = cur.next

while l2:

sum = cur.val + l2.val

cur.val = sum % 10;

cur.next = ListNode(sum // 10)

l2 = l2.next

pre = cur

cur = cur.next

if cur.val == 0:

pre.next = cur.next

return self.reverse(ans)

def getLinkedList(head):

list = []

while head is not None:

list += [str(head.val)]

head = head.next

s = '->'.join(list)

return s

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

head1 = ListNode(6, ListNode(1, ListNode(7)))

head2 = ListNode(2, ListNode(9, ListNode(5)))

print('输入数字1: ' + getLinkedList(head1))

print('输入数字2: ' + getLinkedList(head2))

print('输出二者和: ' + getLinkedList(temp.addLists2(head1,head2)))

例440 程序检查

3.代码实现

import collections

class Solution:

#参数commands为字符串

#返回布尔类型

def check(self, commands):

n = len(commands)

#标签对应的行数

labelIdx = collections.defaultdict(int)

#每个节点的访问状态

visitState = collections.defaultdict(int)

for i in range(n):

if commands[i][0] == 'l':

labelIdx[commands[i][6:]] = i

return self.dfs(0, visitState, labelIdx, commands)

def dfs(self, idx, visitState, labelIdx, commands):

#程序结束

if idx == len(commands):

return False

#程序中有环，可能会死循环

if visitState[idx] == 1:

return True

# visitState = 2, 代表都不会有死循环

if visitState[idx] == 2:

return False

#将idx节点加入栈中

visitState[idx] = 1

flag = False

#停机

if commands[idx][0] == 'h':

visitState[idx] = 2

return False

#跳转

if commands[idx][0] == 'g':

parameters = commands[idx].split()

if len(parameters) == 2:

flag |= self.dfs(labelIdx[parameters[1]], visitState, labelIdx, commands)

else:

flag |= self.dfs(labelIdx[parameters[1]], visitState, labelIdx, commands)

flag |= self.dfs(labelIdx[parameters[2]], visitState, labelIdx, commands)

else:

flag |= self.dfs(idx + 1, visitState, labelIdx, commands)

visitState[idx] = 2

return flag

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

com =["label start","print \"hello world!\"","gotorand start end","print \"good bye\"","halt","label end"]

print('输入命令:' ,com)

print('输出是否可能无限循环:' ,temp.check(com))

例441 特殊回文字符串

3.代码实现

class Solution:

def getpair(self, data):

keywd = {}

for letter in data:

a = letter[0]

b = letter[1]

if a not in keywd.keys():

keywd[a] = []

keywd[a].append(b)

if b not in keywd.keys():

keywd[b] = []

keywd[b].append(a)

return keywd

#参数ambigram为字符串列表

#参数word为字符串

#返回布尔类型

def ispalindrome(self, ambigram, word):

isFalse = True

length = (int)(len(word)/2)

pairkey = self.getpair(ambigram)

for i in range(length):

lst = -1\*(i+1)

substr = word[i]

substr2 =word[lst]

if substr ==substr2:

continue

if substr not in pairkey.keys():

return False

if substr2 not in pairkey.keys():

return False

if (substr in pairkey[substr2]):

continue

for key1 in pairkey[substr2]:

if (substr in pairkey[key1]):

isFalse = False

break

if isFalse:

return False

return True

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

ambigram =["at", "by", "yh", "hn", "mw", "ww"]

word = 'swims'

print('输入交换字母:',ambigram,',输入字符串：',word)

print('输出:' ,temp.ispalindrome(ambigram,word))

例442 数组压缩

3.代码实现

class Solution:

#参数A为数组

#返回整数

def CompressArray(self, A):

leng = len(A[0])

dp = [[] for i in range(leng)]

dp[0] += [0,0,0]

for i in range(1,leng) :

for j in range(3) :

tmp = min(dp[i-1][0] + abs(A[j][i] - A[0][i-1]), dp[i-1][1] + abs(A[j][i] - A[1][i-1]))

tmp = min(tmp, dp[i-1][2] + abs(A[j][i] - A[2][i-1]))

dp[i] +=[tmp]

ans = min(dp[leng - 1][0], min(dp[leng - 1][1], dp[leng - 1][2]))

return ans

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

A = [[1,2,3],[4,5,6],[7,8,9]]

print('输入矩阵:',A)

print('输出最小值:' ,temp.CompressArray(A))

例443 等差矩阵

3.代码实现

class Solution:

def fill(self,i,j,B,row,row\_ok,col,col\_ok):

INF = 0x3f3f3f3f

#如果所在行列都确定了返回即可

if col\_ok[j] and row\_ok[i]:

return

else:

#如果列没有全部确认

if col\_ok[j]==False:

#如果第j列首次确定的数是在i行，或者还没有出现，此时不能计算公差

if col[j]==-1 or col[j]==i:

col[j]=i

else:

#确定第j列

col\_ok[j]=True

#计算出公差

diff = (int)((B[i][j] - B[col[j]][j]) // (i - col[j]))

for r in range(i-1,-1,-1):

if B[r][j]==INF:

B[r][j]=B[r+1][j]-diff

self.fill(r,j,B,row,row\_ok,col,col\_ok)

for r in range(i+1,len(row)):

if B[r][j]==INF:

B[r][j]=B[r-1][j]+diff

self.fill(r,j,B,row,row\_ok,col,col\_ok)

#如果该行还没有全部确认

if row\_ok[i]==False:

#如果i行首次确定数的位置是在j列或者还未出现，此时不能计算公差

if row[i]==-1 or row[i]==j:

row[i]=j

#否则在i行有两个位置已经确定，一个是A[i][j] 一个是A[i][row[i]]

else:

#第i行都可以确定

row\_ok[i]=True

#计算公差

diff = (int)((B[i][j] - B[i][row[i]]) // (j - row[i]))

for c in range(j-1,-1,-1):

if B[i][c]==INF:

B[i][c]=B[i][c+1]-diff

self.fill(i,c,B,row,row\_ok,col,col\_ok)

for c in range(j+1,len(col)):

if B[i][c]==INF:

B[i][c]=B[i][c-1]+diff

self.fill(i,c,B,row,row\_ok,col,col\_ok)

#参数A为矩阵

#参数ask为数组

#返回数组

def getDetermine(self, A, ask):

INF=0x3f3f3f3f

#A矩阵行数

n=len(A)

#A矩阵列数

m=len(A[0])

B=[[INF]\*m for i in range(n)]

#row[i]=j 表示第i行里第一次确定的数是在第j列

row=[-1 for i in range(n)]

#row\_ok[i]=true 表示第i行的数据全部确定

row\_ok=[False for i in range(n)]

#col[j]=i 表示第j列里第一次确定的数是在第i行

col=[-1 for i in range(m)]

#col\_ok[j]=true 表示第j列的数据全部确定

col\_ok=[False for i in range(m)]

for i in range(n):

for j in range(m):

if(A[i][j]!=0):

B[i][j]=A[i][j]

self.fill(i,j,B,row,row\_ok,col,col\_ok)

ans=[]

for i in range(len(ask)):

if B[ask[i][0]][ask[i][1]]!=INF:

ans.append((int)(B[ask[i][0]][ask[i][1]]))

else:

ans.append(-1)

return ans

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

A=[[1,0,3],[0,4,0],[0,0,0]]

ask=[[0,1],[0,0],[1,0],[2,1],[2,2]]

print('输入矩阵:',A)

print('输入位置:',ask)

print('输出位置值:' ,temp.getDetermine(A,ask))

例444 栈集

3.代码实现

class SetOfStacks:

#参数capacity为整数

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

#初始化

self.stacks = []

self.capacity = capacity

#参数v为整数

def push(self, v):

if len(self.stacks) == 0:

self.stacks.append([])

if len(self.stacks[-1]) == self.capacity:

self.stacks.append([])

self.stacks[-1].append(v)

#返回整数

def pop(self):

v = self.stacks[-1].pop()

if len(self.stacks[-1]) is 0:

self.stacks.pop()

return v

if \_\_name\_\_ == '\_\_main\_\_':

temp = SetOfStacks(2)

temp.push(1)

temp.push(2)

temp.push(4)

temp.push(8)

temp.push(16)

print('输入: temp.push(1) temp.push(2) temp.push(4) temp.push(8) temp.push(16)')

print('输出: [' + str(temp.pop()) + ',' + str(temp.pop()) + ',' + str(temp.pop()) + ']')

例445 颜色分类

3.代码实现

class Solution:

#参数colors为整数数组

#参数k为整数

#返回整数数组

def sortColors2(self, colors, k):

self.sort(colors, 1, k, 0, len(colors) - 1)

return colors

def sort(self, colors, color\_from, color\_to, index\_from, index\_to):

if color\_from == color\_to or index\_from == index\_to:

return

color = (color\_from + color\_to) // 2

left, right = index\_from, index\_to

while left <= right:

while left <= right and colors[left] <= color:

left += 1

while left <= right and colors[right] > color:

right -= 1

if left <= right:

colors[left], colors[right] = colors[right], colors[left]

left += 1

right -= 1

self.sort(colors, color\_from, color, index\_from, right)

self.sort(colors, color + 1, color\_to, left, index\_to)

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

A = [3,2,2,1,4]

k = 4

print('输入: A=',A,',k=',k)

print('输出:',temp.sortColors2(A,k))

例446 最长有效括号

3.代码实现

class Solution:

#参数s为字符串

#返回整数

def longestValidParentheses(self, s):

if len(s) <= 1 :

return 0

res = 0

dp = [0 for i in range(len(s))] #初始化

for i in range(len(s) - 2, -1, -1) :

if s[i] == '(' : #如果s[i] = '('，则需要找到右括号和它匹配

j = i + dp[i + 1] + 1

if j < len(s) and s[j] == ')' : #如果没越界且为右括号，那么有dp[i] = dp[i + 1] + 2

dp[i] = dp[i + 1] + 2

if j + 1 < len(s): #将j + 1开头的子串加进来

dp[i] += dp[j + 1]

res = max(res, dp[i])

return res

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

s = ')()())'

print('输入: s=',s)

print('输出:',temp.longestValidParentheses(s))

例447 分糖果

3.代码实现

#采用utf-8编码格式

#参数ratings是一个整数数组

#返回一个整数

class Solution:

def candy(self, ratings):

candynum = [1 for i in range(len(ratings))]

for i in range(1, len(ratings)):

if ratings[i] > ratings[i-1]:

candynum[i] = candynum[i-1] + 1

for i in range(len(ratings)-2, -1, -1):

if ratings[i+1] < ratings[i] and candynum[i+1] >= candynum[i]:

candynum[i] = candynum[i+1] + 1

return sum(candynum)

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

List1 = [2,3,1,1,4]

List2 = [1,4,2,2,3]

print(("输入："+str(List1)))

print(("输出："+str(temp.candy(List1))))

print(("输入："+str(str(List2))))

print(("输出："+str(temp.candy(List2))))

例448 URL 编码

3.代码实现

from urllib.parse import urlencode as urllib\_urlencode

class Solution:

#参数base\_url为字符串

#参数query\_params为查询参数的元组序列

#返回字符串

def urlencode(self, base\_url, query\_params):

if not query\_params:

return base\_url

query\_params.sort()

query\_params = tuple((key, val) for key, val in query\_params)

return base\_url + '?' + urllib\_urlencode(query\_params)

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

base\_url="https://translate.google.cn/"

query\_params=[["sl","en"],["tl","zh-CN"],["text","Hello"],["op","translate"]]

print('输入主机地址:',base\_url)

print('输入查询参数:',query\_params)

print('输出完整的URL:' ,temp.urlencode(base\_url,query\_params))

例449 多字符串查找

3.代码实现

class Solution:

#参数sourceString为字符串

#参数targetStrings为字符串数组

#返回布尔类型数组

def whetherStringsAreSubstrings(self, sourceString, targetStrings):

listin = []

for sr in targetStrings:

index = sourceString.find(sr)

if index != -1:

listin.append(True)

else:

listin.append(False)

return listin

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

sourceString="abc"

targetStrings=["ab","cd"]

print('输入源字符串:',sourceString)

print('输入目标字符串:',targetStrings)

print('输出是否为子串:' ,temp.whetherStringsAreSubstrings(sourceString,targetStrings))

例450 最大订单

3.代码实现

class Solution:

#参数onHand为已有奶精过期天数，整数数组

#参数supplier为可提供的奶精过期天数，整数数组

#参数demand为职员最大需求，整数

#返回需要订购的数量，整数

def check(self, onHand, supplier, demand, order):

m = len(onHand)

n = len(supplier)

onHnadIndex = 0

supplierIndex = n - order

for i in range(m + order):

if supplierIndex < n and ( supplier[supplierIndex] <= onHand[onHnadIndex] or onHnadIndex == m):

if supplier[supplierIndex] < i // demand:

return 0

supplierIndex += 1

else:

if onHand[onHnadIndex] < i // demand:

return 0

onHnadIndex += 1

return 1

def stockLounge(self, onHand, supplier, demand):

m = len(onHand)

n = len(supplier)

onHand=sorted(onHand)

supplier=sorted(supplier)

for i in range(m):

if onHand[i] < i // demand:

return -1

left = 0

right = n

while left + 1 < right:

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

if self.check(onHand, supplier, demand, mid):

left = mid

else:

right = mid - 1

if self.check(onHand, supplier, demand, right):

return right

else:

return left

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

onHand=[0,2,2]

supplier=[2,0,0]

demand=2

print('输入:',onHand,',',supplier,',',demand)

print('输出:',temp.stockLounge(onHand, supplier, demand))

例451 最长字符串链

3.代码实现

class Solution:

#参数words为单词列表

#返回整数

"""

def pre\_word(self, a, b):

if len(a) + 1 != len(b):

return False

i = 0

j = 0

while i < len(a) and j < len(b):

if a[i] == b[j]:

i += 1

j += 1

if(i == len(a)):

return True

return False

def longestStrChain(self, words):

dp = [0 for i in range(len(words))]

ans = 0

words = sorted(words, key=lambda x: len(x))

for i in range(len(words)):

for j in range(i):

if self.pre\_word(words[j], words[i]):

dp[i] = int(max(dp[i], dp[j] + 1))

ans = int(max(ans, dp[i]))

return ans + 1

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

words=["a","b","ba","bca","bda","bdca"]

print('输入:',words)

print('输出:',temp.longestStrChain(words))

例452 地图跳跃

3.代码实现

class Solution:

#参数arr为地图矩阵

#返回整数

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

self.vis = [[0 for i in range(108)] for i in range(105)];

self.m = 0;

def mapJump(self, arr):

n = len(arr);

l = 0;

r = 100000;

while l <= r:

self.m = (l + r) >> 1;

for i in range(0,len(arr)):

for j in range(0,len(arr)):

self.vis[i][j] = 0;

self.dfs(0, 0, arr);

if self.vis[n - 1][n - 1] == 1:

ans = self.m;

r = self.m - 1;

else:

l = self.m + 1;

return ans;

def dfs(self, x, y, arr):

dx = [0, 0, 1, -1];

dy = [1, -1, 0, 0];

self.vis[x][y] = 1;

for i in range(0, 4):

sx = x + dx[i];

sy = y + dy[i];

if sx >= len(arr) or sy >= len(arr) or sx < 0 or sy < 0:

continue;

if abs(arr[x][y] - arr[sx][sy]) > self.m or self.vis[sx][sy] == 1:

continue;

self.dfs(sx, sy, arr);

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

arr=[[1,5],[6,2]]

print('输入:',arr)

print('输出:',temp.mapJump(arr))

例453 查找最大因子

3.代码实现

class Solution:

#参数A为数组

#参数k为整数

#返回整数

"""

def FindDivisor(self, A, k):

n = len(A)

left = 1

right = 1

d = 0

for i in range(n):

right = int(max(right, A[i]))

while (left < right) :

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

sum\_d = 0

for i in range(n):

sum\_d = sum\_d + int(A[i] / mid)

if A[i] % mid != 0:

sum\_d += 1

if sum\_d >= k :

d = mid

left = mid + 1

else:

right = mid

return d

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

A=[1,2,3,4,5]

k=6

print('输入:A=',A,',k=',k)

print('输出:',temp.FindDivisor(A,k))

例454 矩阵斜线上元素相同

3.代码实现

class Solution:

#参数matrix为矩阵

#返回值为布尔类型

def judgeSame(self, matrix):

n = len(matrix)

for i in range(1, n):

for j in range(1, n):

if (matrix[i][j] != matrix[i - 1][j - 1]):

return False

return True

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

matrix=[[1,2,3],[5,1,2], [6,5,1]]

print('输入:A=',matrix)

print('输出:',temp.judgeSame(matrix))

例455 最大连通面积

3.代码实现

from collections import deque

class Solution:

#参数matrix为矩阵

#返回整数

def maxArea(self, matrix):

m, n = len(matrix), len(matrix[0])

res = float('-inf')

size = {}

index = 0

#1连通索引和的面积

for i in range(m):

for j in range(n):

if matrix[i][j] == 1 and (i, j) not in size:

index += 1

count = self.bfs(matrix, size, index, i, j)

#遍历每 0 以找到最大连接区域

for i in range(m):

for j in range(n):

if matrix[i][j] == 0:

area = 1

seenIsland = set()

for dx, dy in [(0, 1), (1, 0), (-1, 0), (0, -1)]:

nx = i + dx

ny = j + dy

if not 0 <= nx < m or not 0 <= ny < n:

continue

if (nx, ny) not in size:

continue

index, count = size[(nx, ny)]

if index in seenIsland:

continue

seenIsland.add(index)

area += count

res = max(res, area)

return res if res != float('-inf') else m \* n

def bfs(self, matrix, size, index, x, y):

m, n = len(matrix), len(matrix[0])

q = deque([(x, y)])

vis = set([(x, y)])

count = 0

while q:

x, y = q.popleft()

count += 1

for dx, dy in [(1, 0), (0, 1), (-1, 0), (0, -1)]:

nx = x + dx

ny = y + dy

if not 0 <= nx < m or not 0 <= ny < n:

continue

if (nx, ny) in vis:

continue

if matrix[nx][ny] == 1:

q.append((nx, ny))

vis.add((nx, ny))

for x, y in vis:

size[(x, y)] = (index, count)

return count

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

matrix=[[0,1],[1,0]]

print('输入:A=',matrix)

print('输出:',temp.maxArea(matrix))

例456 小括号匹配

3.代码实现

class Solution:

#参数string为字符串

#返回布尔类型

def matchParentheses(self, string):

stack = []

for c in string:

if c == '(':

stack.append(c)

else:

if not stack:

return False

if stack[-1] == '(':

stack.pop()

return True

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

string='()'

print('输入:string=',string)

print('输出:',temp.matchParentheses(string))

例457 通用子数组个数

3.代码实现

class Solution:

#参数array为数组

#返回整数

def subarrays(self, array):

size = len(array)

#记录当前连续2，4的个数

count\_2 = 0

count\_4 = 0

#存放连续2，4个数的数组

queue = []

for i in range(size):

if array[i] == 4:

if i > 0 and array[i-1] == 2:

queue.append(count\_2)

count\_2 = 0

count\_4 += 1

if array[i] == 2:

if i > 0 and array[i-1] == 4:

queue.append(count\_4)

count\_4 = 0

count\_2 += 1

#处理最后一段连续2或4

if array[size-1] == 4:

queue.append(count\_4)

else:

queue.append(count\_2)

#相邻的两个数取min累加到结果

result = 0

for i in range(1, len(queue)):

result += min(queue[i], queue[i-1])

return result

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

array=[4,4,2,2,4,2]

print('输入: array=',array)

print('输出:',temp.subarrays(array))

例458 最大非负子序和

3.代码实现

class Solution:

#参数A为数组

#返回整数

def maxNonNegativeSubArray(self, A):

#A数组长度

n = len(A)

#maxSubArraySum[i] 表示以A[i]结尾的最大非负子序和

#若maxSubArraySum[i]为-1 ，表示A[i] 为负值

#因为maxSubArraySum[i]只与maxSubArraySum[i-1]，A[i]有关，可以只使用一个变量lastIndexSubArraySum记录maxSubArraySum[i-1]

lastIndexSubArraySum = -1

#初始化边界条件

#记录0位置的答案

if A[0] >= 0:

lastIndexSubArraySum = A[0]

maxSubArraySumAnswer = -1

maxSubArraySumAnswer = lastIndexSubArraySum

for i in range(1, n):

#用nowIndexSubArraySum来代替maxSubArraySum[i]

nowIndexSubArraySum = -1

#如果A[i]为非负整数，计算A[i]结尾的最大非负子序和

if A[i] >= 0:

#maxSubArraySum[i-1]为-1 表明A[i-1]为负数，从i位置重新开始一段新的子数组

#用lastIndexSubArraySum记录maxSubArraySum[i-1]

#只用maxSubArraySum[i-1]即可判断

if lastIndexSubArraySum == -1:

nowIndexSubArraySum = A[i]

#maxSubArraySum[i-1]不是-1 表明A[i-1]为非负整数，将A[i]接在A[i-1]的子数组的后面

#用lastIndexSubArraySum记录maxSubArraySum[i-1]

#只用maxSubArraySum[i-1]即可判断

else:

nowIndexSubArraySum = lastIndexSubArraySum + A[i] #更新

maxSubArraySumAnswer = max(maxSubArraySumAnswer, nowIndexSubArraySum)

#更新lastIndexSubArraySum

lastIndexSubArraySum = nowIndexSubArraySum

#如果maxSubArraySumAnswer还是-1 说明整个A数组都是负数

return maxSubArraySumAnswer

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

A=[1,2,-3,4,5,-6]

print('输入: A=',A)

print('输出:',temp.maxNonNegativeSubArray(A))

例459 最短休息日

3.代码实现

class Solution:

#参数company为数组

#参数gym为数组

#返回整数

def minimumRestDays(self, company, gym):

dp = [[float('inf')] \* 3 for \_ in range(len(company))]

dp[0][0] = 1

if company[0]:

dp[0][1] = 0

if gym[0]:

dp[0][2] = 0

for i in range(1, len(company)):

dp[i][0] = min(dp[i - 1][0], dp[i - 1][1], dp[i - 1][2]) + 1

if company[i]:

dp[i][1] = min(dp[i - 1][0], dp[i - 1][2])

if gym[i]:

dp[i][2] = min(dp[i - 1][0], dp[i - 1][1])

return min(dp[len(company) - 1][0], dp[len(company) - 1][1], dp[len(company) - 1][2])

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

company=[1,1,0,0]

gym=[0,1,1,0]

print('输入: company=',company,',gym=',gym)

print('输出:',temp.minimumRestDays(company,gym))

例460 括号得分

3.代码实现

class Solution:

#参数S为字符串，包括"("和")"

#返回整数

def ParenthesesScore(self, S):

stack = []

answer = 0

for i, c in enumerate(S):

if c == '(':

stack.append([i, 0])

elif c == ')':

\_, value = stack.pop()

if value == 0:

value += 1

else:

value \*= 2

if stack:

stack[-1][1] += value

else:

answer += value

return answer

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

S="(())"

print('输入: S=',S)

print('输出:',temp.ParenthesesScore(S))

例461 双色塔

3.代码实现

import math

class Solution:

#参数red为红色石头数量

#参数green为绿色石头数量

#返回整数

def twoColorsTower(self, red, green):

MOD = int(1e9) + 7

dp = [[0] \* (red + 1) for \_ in range(2)]

dp[0][0] = 1

answer = 0

for i in range(1, int(math.ceil(math.sqrt(red + green) \* 2)) + 1):

for j in range(red + 1):

dp[i % 2][j] = 0

if j - i >= 0 and red - j >= 0:

dp[i % 2][j] += dp[(i - 1) % 2][j - i]

dp[i % 2][j] %= MOD

num\_green\_used = (i + 1) \* i // 2 - j

if green - num\_green\_used >= 0:

dp[i % 2][j] += dp[(i - 1) % 2][j]

dp[i % 2][j] %= MOD

if sum(dp[i % 2]) > 0:

answer = sum(dp[i % 2]) % MOD

return answer

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

red=4

green=6

print('输入: red=',red,',green=',green)

print('输出:',temp.twoColorsTower(red,green))

例462 考试策略

3.代码实现

class Solution:

#参数p为部分花费时间数组

#参数part为部分得分数组

#参数f为全部花费时间数组

#参数full为全部得分数组

#返回整数得分

def exam(self, p, part, f, full):

dp = [0 for j in range(120 + 1)]

dp[0] = 0

for i in range(len(part)):

for j in range(120, -1, -1):

if j - p[i] >= 0:

dp[j] = max(dp[j], dp[j - p[i]] + part[i])

if j - f[i] >= 0:

dp[j] = max(dp[j], dp[j - f[i]] + full[i])

return max(dp)

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

p=[20,50,100,5]

part=[20,30,60,3]

f=[100,80,110,10]

full=[60,55,88,6]

print('输入:p=',p,',part=',part,',f=',f,',full=',full)

print('输出:',temp.exam(p, part, f, full))

例463 移动车棚

3.代码实现

class Solution:

#参数stops为停车位置数组

#参数k为车棚需要覆盖的大小

#返回整数最小值

def calculate(self, stops, k):

#车的总数

n = len(stops)

#对停车的位置进行升排序

stops = sorted(stops)

#车棚要覆盖所有的车

if (n == k):

return stops[n - 1] - stops[0] + 1

#将车棚长度初始化为第 1 辆车和第 k+1 辆车 - 1 的距离长度

shed = stops[k] - stops[0]

#利用双指针代表车棚的左部和右部

left = 1

right = k + 1

#移动车棚，找到满足条件的车棚长度

while right < n:

shed = max(stops[right] - stops[left], shed)

left += 1

right += 1

return shed

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

stops=[7,3,6,1,8]

k=3

print('输入: stops=',stops,',k=',k)

print('输出:',temp.calculate(stops,k))

例464 另一个祖玛游戏

3.代码实现

class Solution:

#参数s为输入序列

#参数k为相邻个数

#返回消融后序列

def zumaGaming(self, s, k):

stack = []

pairs = []

for c in s:

if pairs and pairs[-1][0] == c:

pairs[-1][1] += 1

else:

pairs.append([c, 1])

for c, num in pairs:

if stack and stack[-1][0] == c:

stack[-1][1] += num

else:

stack.append([c, num])

if stack and stack[-1][1] >= k:

stack[-1][1] %= k

if stack[-1][1] == 0:

stack.pop()

answer = []

for c, num in stack:

answer.extend([c] \* num)

return ''.join(answer)

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

s="abbaca"

k=2

print('输入: s=',s,',k=',k)

print('输出:',temp.zumaGaming(s,k))

例465 罗马数字转整数

3.代码实现

#采用utf-8编码格式

#参数s是一个字符串

#返回一个整数值

class Solution:

def romanToInt(self, s):

ROMAN = {

'I': 1,

'V': 5,

'X': 10,

'L': 50,

'C': 100,

'D': 500,

'M': 1000

}

if s == "":

return 0

index = len(s) - 2

sum = ROMAN[s[-1]]

while index >= 0:

if ROMAN[s[index]] < ROMAN[s[index + 1]]:

sum -= ROMAN[s[index]]

else:

sum += ROMAN[s[index]]

index -= 1

return sum

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

string1 = "DCXXI"

string2 = "XX"

print(("输入："+string1))

print(("输出："+str(temp.romanToInt(string1))))

print(("输入："+string2))

print(("输出："+str(temp.romanToInt(string2))))

例466 两个整数相除

3.代码实现

#采用utf-8编码格式

class Solution(object):

def divide(self, dividend, divisor):

INT\_MAX = 2147483647

if divisor == 0:

return INT\_MAX

neg = dividend > 0 and divisor < 0 or dividend < 0 and divisor > 0

a, b = abs(dividend), abs(divisor)

ans, shift = 0, 31

while shift >= 0:

if a >= b << shift:

a -= b << shift

ans += 1 << shift

shift -= 1

if neg:

ans = - ans

if ans > INT\_MAX:

return INT\_MAX

return ans

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

x1 = 100

x2 = 10

print(("输入："+str(x1)+" "+str(x2)))

print(("输出："+str(temp.divide(x1,x2))))

例467 滑动窗口的最大值

3.代码实现

#采用utf-8编码格式

#参数nums是一个整数数组

#参数k是一个整数

#返回值是数组中每个窗口内的最大值

from collections import deque

class Solution:

def maxSlidingWindow(self, nums, k):

if not nums or not k:

return []

dq = deque([])

for i in range(k - 1):

self.push(dq, nums, i)

result = []

for i in range(k - 1, len(nums)):

self.push(dq, nums, i)

result.append(nums[dq[0]])

if dq[0] == i - k + 1:

dq.popleft()

return result

def push(self, dq, nums, i):

while dq and nums[dq[-1]] < nums[i]:

dq.pop()

dq.append(i)

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

List1 = [2,6,5,3,1,8]

nums1 = 2

print(("输入："+str(List1)+" "+str(nums1)))

print(("输出："+str(temp.maxSlidingWindow(List1,nums1))))

例468 镜像数字

3.代码实现

#参数num是一个字符串  
#返回一个布尔值，判断这个数字是否是镜像的  
class Solution:  
 def isStrobogrammatic(self, num):  
 map = {'0': '0', '1': '1', '6': '9', '8': '8', '9': '6'}  
 i, j = 0, len(num) - 1  
 while i <= j:  
 if not num[i] in map or map[num[i]] != num[j]:  
 return False  
 i, j = i + 1, j - 1  
 return True  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 num = "68"  
 #创建对象  
 solution = Solution()  
 print("输入：", num)  
 print("输出：", solution.isStrobogrammatic(num))

例469 直方图中最大矩形面积

3.代码实现

#参数为一个整数列表  
#返回在柱状图中长方形的最大面积  
class Solution:  
 def largestRectangleArea(self, heights):  
 indices\_stack = []  
 area = 0  
 for index, height in enumerate(heights + [0]):  
 while indices\_stack and heights[indices\_stack[-1]] >= height:  
 popped\_index = indices\_stack.pop()  
 left\_index = indices\_stack[-1] if indices\_stack else -1  
 width = index - left\_index - 1  
 area = max(area, width \* heights[popped\_index])  
 indices\_stack.append(index)  
 return area  
#主函数  
if \_\_name\_\_=="\_\_main\_\_":  
 heights=[2,1,5,6,2,3]  
 #创建对象  
 solution=Solution()  
 print("输入每个直方图高度：",heights)  
 print("找到直方图最大面积：",solution.largestRectangleArea(heights))

例470 最长回文子串

3.代码实现

#采用utf-8编码格式

#参数s是一个输入的字符串

#返回值是最长的回文子串

class Solution:

def longestPalindrome(self, s):

if not s:

return ""

longest = ""

for middle in range(len(s)):

sub = self.find\_palindrome\_from(s, middle, middle)

if len(sub) > len(longest):

longest = sub

sub = self.find\_palindrome\_from(s, middle, middle + 1)

if len(sub) > len(longest):

longest = sub

return longest

def find\_palindrome\_from(self, string, left, right):

while left >= 0 and right < len(string) and string[left] == string[right]:

left -= 1

right += 1

return string[left + 1:right]

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

string1 = "abcdedcb"

string2 = "qwerfdfdfg"

print(("输入："+string1))

print(("输出："+str(temp.longestPalindrome(string1))))

print(("输入："+string2))

print(("输出："+str(temp.longestPalindrome(string2))))

例471 乱序字符串

3.代码实现

#采用utf-8编码格式

#参数strs是一个字符串

#返回一个字符串

class Solution:

def anagrams(self, strs):

dict = {}

for word in strs:

sortedword = ''.join(sorted(word))

if sortedword not in dict:

dict[sortedword] = [word]

else:

dict[sortedword] += [word]

res = []

for item in dict:

if len(dict[item]) >= 2:

res += dict[item]

return res

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

List1 = ["abcd","bcad","dabc","etc"]

List2 = ["mkji","ijkm","kjim","imjk"]

print(("输入："+str(List1)))

print(("输出："+str(temp.anagrams(List1))))

print(("输入："+str(List2)))

print(("输出："+str(temp.anagrams(List2))))

例472 交叉字符串

3.代码实现

#参数s1、s2、s3是三个描述中提到的字符串

#返回值：如果s3是由s1和s2的交叉形成的，则返回True；如果不是，则返回False

#可以使用[[True]\*m表示范围(n)]中的i分配一个n行m列矩阵。

class Solution:

def isInterleave(self, s1, s2, s3):

if s1 is None or s2 is None or s3 is None:

return False

if len(s1) + len(s2) != len(s3):

return False

interleave = [[False] \* (len(s2) + 1) for i in range(len(s1) + 1)]

interleave[0][0] = True

for i in range(len(s1)):

interleave[i + 1][0] = s1[:i + 1] == s3[:i + 1]

for i in range(len(s2)):

interleave[0][i + 1] = s2[:i + 1] == s3[:i + 1]

for i in range(len(s1)):

for j in range(len(s2)):

interleave[i + 1][j + 1] = False

if s1[i] == s3[i + j + 1]:

interleave[i + 1][j + 1] = interleave[i][j + 1]

if s2[j] == s3[i + j + 1]:

interleave[i + 1][j + 1] |= interleave[i + 1][j]

return interleave[len(s1)][len(s2)]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s1 = "aabcc"

s2 = "dbbca"

s3 = "aadbbcbcac"

print("数组s1：", s1)

print("数组s2：", s2)

print("数组s3：", s3)

solution = Solution()

print("数组是否交叉：", solution.isInterleave(s1, s2, s3))

例473 回文链表

3.代码实现

#创建链表  
#参数head是一个链表节点  
#返回一个布尔值  
class ListNode(object):  
 def \_\_init\_\_(self, val, next=None):  
 self.val = val  
 self.next = next  
class Solution:  
 def isPalindrome(self, head):  
 if head is None:  
 return True  
 fast = slow = head  
 while fast.next and fast.next.next:  
 slow = slow.next  
 fast = fast.next.next  
 p, last = slow.next, None  
 while p:  
 next = p.next  
 p.next = last  
 last, p = p, next  
 p1, p2 = last, head  
 while p1 and p1.val == p2.val:  
 p1, p2 = p1.next, p2.next  
 p, last = last, None  
 while p:  
 next = p.next  
 p.next = last  
 last, p = p, next  
 slow.next = last  
 return p1 is None  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 node1 = ListNode(1)  
 node2 = ListNode(2)  
 node3 = ListNode(1)  
 node1.next = node2  
 node2.next = node3  
 #创建对象  
 solution = Solution()  
 print("初始链表：", [node1.val, node2.val, node3.val])  
 print("最终结果：", solution.isPalindrome(node1))

例474 链表插入排序

3.代码实现

#定义链表  
#参数head是连接链表的第一个节点  
#返回值是连接链表的头节点  
class ListNode(object):  
 def \_\_init\_\_(self, val, next=None):  
 self.val = val  
 self.next = next  
class Solution:  
 def insertionSortList(self, head):  
 dummy = ListNode(0)  
 while head:  
 temp = dummy  
 next = head.next  
 while temp.next and temp.next.val < head.val:  
 temp = temp.next  
 head.next = temp.next  
 temp.next = head  
 head = next  
 return dummy.next  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 node1 = ListNode(1)  
 node2 = ListNode(3)  
 node3 = ListNode(2)  
 node4 = ListNode(0)  
 node1.next = node2  
 node2.next = node3  
 node3.next = node4  
 list1 = []  
 #创建对象  
 solution = Solution()  
 print("初始链表：", [node1.val, node2.val, node3.val, node4.val])  
 newlist = solution.insertionSortList(node1)  
 while (newlist):  
 list1.append(newlist.val)  
 newlist = newlist.next  
 print("插入排序后的链表：", list1)

例475 具有最大平均数的子树

3.代码实现

#树的定义

class TreeNode:

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

self.val = val

self.left, self.right = None, None

class Solution:

#参数root是一个二叉树的根节点

#返回值是最大平均值子树的根节点

average, node = 0, None

def findSubtree2(self, root):

self.helper(root)

return self.node

def helper(self, root):

if root is None:

return 0, 0

left\_sum, left\_size = self.helper(root.left)

right\_sum, right\_size = self.helper(root.right)

sum, size = left\_sum + right\_sum + root.val, left\_size + right\_size + 1

if self.node is None or sum \* 1.0 / size > self.average:

self.node = root

self.average = sum \* 1.0 / size

return sum, size

def printTree(root):

res = []

if root is None:

print(res)

queue = []

queue.append(root)

while len(queue) != 0:

tmp = []

length = len(queue)

for i in range(length):

r = queue.pop(0)

if r.left is not None:

queue.append(r.left)

if r.right is not None:

queue.append(r.right)

tmp.append(r.val)

res.append(tmp)

return (res)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

root = TreeNode(1)

root.left = TreeNode(-5)

root.right = TreeNode(11)

root.left.left = TreeNode(1)

root.left.right = TreeNode(2)

root.right.left = TreeNode(4)

root.right.right = TreeNode(-2)

solution = Solution()

print("给定二叉树：", printTree(root))

print("最大平均值的子树：", printTree(solution.findSubtree2(root)))

例476 高度平衡二叉树

3.代码实现

#树的定义

#参数root是二叉树的根节点

#返回值是一个布尔值，如果二叉树是平衡树则返回True，否则返回False

class TreeNode:

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

self.val = val

self.left, self.right = None, None

class Solution:

def isBalanced(self, root):

balanced, \_ = self.validate(root)

return balanced

def validate(self, root):

if root is None:

return True, 0

balanced, leftHeight = self.validate(root.left)

if not balanced:

return False, 0

balanced, rightHeight = self.validate(root.right)

if not balanced:

return False, 0

return abs(leftHeight - rightHeight) <= 1, max(leftHeight, rightHeight) + 1

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

#树A

root = TreeNode(3)

root.left = TreeNode(9)

root.right = TreeNode(20)

root.right.left = TreeNode(15)

root.right.right = TreeNode(7)

#树B

root1 = TreeNode(3)

root1.right = TreeNode(20)

root1.right.left = TreeNode(15)

root1.right.right = TreeNode(7)

solution = Solution()

print("树A是否平衡：", solution.isBalanced(root))

print("树B是否平衡：", solution.isBalanced(root1))

例477 主元素

3.代码实现

#参数nums是一个整数数组

#返回找到一个主元素

class Solution:

def majorityNumber(self, nums):

key, count = None, 0

for num in nums:

if key is None:

key, count = num, 1

else:

if key == num:

count += 1

else:

count -= 1

if count == 0:

key = None

return key

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

nums1 = [2,2,2,3,3,3,3]

nums2 = [1,2,3,4]

print ("输入数组："+"[2,2,2,3,3,3,3]"+"\n输出："+str(temp.majorityNumber(nums1)))

print ("输入数组："+"[1,2,3,4]"+"\n输出："+str(temp.majorityNumber(nums2)))

例478 单词矩阵

3.代码实现

class TrieNode:

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

self.children = {}

self.is\_word = False

self.word\_list = []

class Trie:

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

self.root = TrieNode()

def add(self, word):

node = self.root

for c in word:

if c not in node.children:

node.children[c] = TrieNode()

node = node.children[c]

node.word\_list.append(word)

node.is\_word = True

def find(self, word):

node = self.root

for c in word:

node = node.children.get(c)

if node is None:

return None

return node

def get\_words\_with\_prefix(self, prefix):

node = self.find(prefix)

return [] if node is None else node.word\_list

def contains(self, word):

node = self.find(word)

return node is not None and node.is\_word

class Solution:

#参数words代表没有重复的一系列单词集合

#返回所有单词矩阵

def wordSquares(self, words):

trie = Trie()

for word in words:

trie.add(word)

squares = []

for word in words:

self.search(trie, [word], squares)

return squares

def search(self, trie, square, squares):

n = len(square[0])

curt\_index = len(square)

if curt\_index == n:

squares.append(list(square))

return

#修剪，可以删除它，但会比较慢

for row\_index in range(curt\_index, n):

prefix = ''.join([square[i][row\_index] for i in range(curt\_index)])

if trie.find(prefix) is None:

return

prefix = ''.join([square[i][curt\_index] for i in range(curt\_index)])

for word in trie.get\_words\_with\_prefix(prefix):

square.append(word)

self.search(trie, square, squares)

square.pop()

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

word = ["area", "lead", "wall", "lady", "ball"]

print("单词序列：", word)

solution = Solution()

print("构成的单词矩阵：", solution.wordSquares(word))

例479 电话号码的字母组合

3.代码实现

#采用utf-8编码格式

#输出电话号码对应所有可能的字符串

#可以递归或直接模拟

import copy

class Solution(object):

def letterCombinations(self, digits):

chr = ["", "", "abc", "def", "ghi", "jkl", "mno", "pqrs", "tuv", "wxyz"]

res = []

for i in range(0, len(digits)):

num = int(digits[i])

tmp = []

for j in range(0, len(chr[num])):

if len(res):

for k in range(0, len(res)):

tmp.append(res[k] + chr[num][j])

else:

tmp.append(str(chr[num][j]))

res = copy.copy(tmp)

return res

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

string1 = "3"

string2 = "5"

print(("输入："+string1))

print(("输出："+str(temp.letterCombinations(string1))))

print(("输入："+string2))

print(("输出："+str(temp.letterCombinations(string2))))

例480 会议室

3.代码实现

#参数intervals是一个会议室时间间隔的数组

#返回值是所需的最小会议室数量

class Interval(object):

def \_\_init\_\_(self, start, end):

self.start = start

self.end = end

class Solution:

def minMeetingRooms(self, intervals):

points = []

for interval in intervals:

points.append((interval.start, 1))

points.append((interval.end, -1))

meeting\_rooms = 0

ongoing\_meetings = 0

for \_, delta in sorted(points):

ongoing\_meetings += delta

meeting\_rooms = max(meeting\_rooms, ongoing\_meetings)

return meeting\_rooms

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

node1 = Interval(0, 30)

node2 = Interval(5, 10)

node3 = Interval(15, 20)

print("会议时间间隔：", [[node1.start, node1.end], [node2.start, node2.end], [node3.start, node3.end]])

intervals = [node1, node2, node3]

solution = Solution()

print("最小的会议室数量：", solution.minMeetingRooms(intervals))

例481 无重叠区间

3.代码实现

#采用utf-8编码格式

import sys

class Solution:

def eraseOverlapIntervals(self, intervals):

ans = 0

end = -sys.maxsize

for i in sorted(intervals, key=lambda i: i[-1]):

if i[0] >= end:

end = i[-1]

else:

ans += 1

return ans

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

List1 = [ [1,2], [2,3], [3,4], [1,3] ]

List2 = [ [1,2], [1,2], [1,2] ]

print(("输入："+str(List1)))

print(("输出："+str(temp.eraseOverlapIntervals(List1))))

print(("输入："+str(str(List2))))

print(("输出："+str(temp.eraseOverlapIntervals(List2))))

例482 表达式求值

3.代码实现

#字符串列表  
#返回一个整数  
class Solution:  
 def evaluateExpression(self, expression):  
 if expression is None or len(expression) == 0:  
 return 0  
 integers = []  
 symbols = []  
 for c in expression:  
 if c.isdigit():  
 integers.append(int(c))  
 elif c == "(":  
 symbols.append(c)  
 elif c == ")":  
 while symbols[-1] != "(":  
 self.calculate(integers, symbols)  
 symbols.pop()  
 else:  
 if symbols and symbols[-1] != "(" and self.get\_level(c) >= self.get\_level(symbols[-1]):  
 self.calculate(integers, symbols)  
 symbols.append(c)  
 while symbols:  
 print(integers, symbols)  
 self.calculate(integers, symbols)  
 if len(integers) == 0:  
 return 0  
 return integers[0]  
 def get\_level(self, c):  
 if c == "+" or c == "-":  
 return 2  
 if c == "\*" or c == "/":  
 return 1  
 return sys.maxsize  
 def calculate(self, integers, symbols):  
 if integers is None or len(integers) < 2:  
 return False  
 after = integers.pop()  
 before = integers.pop()  
 symbol = symbols.pop()  
 if symbol == "-":  
 integers.append(before - after)  
 elif symbol == "+":  
 integers.append(before + after)  
 elif symbol == "\*":  
 integers.append(before \* after)  
 elif symbol == "/":  
 integers.append(before // after)  
 return True  
#主函数  
if \_\_name\_\_=="\_\_main\_\_":  
 str="(2\*6-(23+7)/(1+2))"  
 num=["2", "\*", "6", "-", "(", "23", "+", "7", ")", "/","(", "1", "+", "2", ")"]  
 #创建对象  
 solution=Solution()  
 print("输入表达式：", str)  
 print("其表达式对应的数组：", num)  
 print("表达式的值：",solution.evaluateExpression(num))

例483 翻转游戏

3.代码实现

#参数s是一个给定的字符串

#返回值是一个布尔值，如果能够保证先手胜利则返回True

class Solution:

memo = {}

def canWin(self, s):

if s in self.memo:

return self.memo[s]

for i in range(len(s) - 1):

if s[i:i + 2] == '++':

tmp = s[:i] + '--' + s[i + 2:]

flag = self.canWin(tmp)

self.memo[tmp] = flag

if not flag:

return True

return False

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

s = "++++"

print("s是：", s)

solution = Solution()

print("是否可以赢：", solution.canWin(s))

例484 迷宫

3.代码实现

DIRECTIONS = [(1, 0), (-1, 0), (0, -1), (0, 1)]

class Solution(object):

def hasPath(self, maze, start, destination):

if not maze:

return False

visited, self.ans = {(start[0], start[1])}, False

self.dfs\_helper(maze, start[0], start[1], destination, visited)

return self.ans

def dfs\_helper(self, maze, x, y, destination, visited):

if self.ans or self.is\_des(x, y, destination):

self.ans = True

return

for dx, dy in DIRECTIONS:

new\_x, new\_y = x, y

while self.is\_valid(maze, new\_x + dx, new\_y + dy):

new\_x += dx

new\_y += dy

coor = (new\_x, new\_y)

if coor not in visited:

visited.add(coor)

self.dfs\_helper(maze, new\_x, new\_y, destination, visited)

def is\_valid(self, maze, x, y):

row, col = len(maze), len(maze[0])

return 0 <= x < row and 0 <= y < col and maze[x][y] == 0

def is\_des(self, x, y, destination):

return x == destination[0] and y == destination[1]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

maze = [[0, 0, 1, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 1, 0], [1, 1, 0, 1, 1], [0, 0, 0, 0, 0]]

start = [0, 4]

destination = [4, 4]

print("迷宫：", maze)

print("初始地点:", start)

print("终点：", destination)

solution = Solution()

print("是否可以走出迷宫:", solution.hasPath(maze, start, destination))

例485 摆动排序问题

3.代码实现

#参数nums是一个整数数组

#返回整数数组

class Solution:

def wiggleSort(self, nums):

if not nums:

return

for i in range(1, len(nums)):

should\_swap = nums[i] < nums[i - 1] if i % 2 else nums[i] > nums[i - 1]

if should\_swap:

nums[i], nums[i - 1] = nums[i - 1], nums[i]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

nums = [3, 5, 2, 1, 6, 4]

print("初始数组：", nums)

solution = Solution()

solution.wiggleSort(nums)

print("结果：", nums)

例486 排颜色

3.代码实现

#参数colors是一个整数数组

#参数k是一个整数

#返回排序后的数组

class Solution:

def sortColors2(self, colors, k):

self.sort(colors, 1, k, 0, len(colors) - 1)

def sort(self, colors, color\_from, color\_to, index\_from, index\_to):

if color\_from == color\_to or index\_from == index\_to:

return

color = (color\_from + color\_to) // 2

left, right = index\_from, index\_to

while left <= right:

while left <= right and colors[left] <= color:

left += 1

while left <= right and colors[right] > color:

right -= 1

if left <= right:

colors[left], colors[right] = colors[right], colors[left]

left += 1

right -= 1

self.sort(colors, color\_from, color, index\_from, right)

self.sort(colors, color + 1, color\_to, left, index\_to)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

colors = [3, 2, 2, 1, 4]

k = 4

print("初始对象和颜色种类：", colors,k)

solution = Solution()

solution.sortColors2(colors, k)

print("结果：", colors)

例487 颜色分类

3.代码实现

#参数nums是一个整数数组，包括0、1、2

#返回排序后的数组

class Solution:

def sortColors(self, A):

left, index, right = 0, 0, len(A) - 1

#注意index < right不正确

while index <= right:

if A[index] == 0:

A[left], A[index] = A[index], A[left]

left += 1

index += 1

elif A[index] == 1:

index += 1

else:

A[right], A[index] = A[index], A[right]

right -= 1

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

A = [1, 0, 1, 2]

print("初始数组：", A)

solution = Solution()

solution.sortColors(A)

print("结果：", A)

例488 简化路径

3.代码实现

#path是一个字符串  
#返回一个字符串  
class Solution:  
 def simplifyPath(self, path):  
 stack = []  
 i = 0  
 res = ''  
 while i < len(path):  
 end = i+1  
 while end < len(path) and path[end] != "/":  
 end += 1  
 sub=path[i+1:end]  
 if len(sub) > 0:  
 if sub == "..":  
 if stack != []: stack.pop()  
 elif sub != ".":  
 stack.append(sub)  
 i = end  
 if stack == []: return "/"  
 for i in stack:  
 res += "/"+i  
 return res  
#主函数  
if \_\_name\_\_=="\_\_main\_\_":  
 path="/home/"  
 #创建对象  
 solution=Solution()  
 print("输入路径：",path)  
 print("路径简化后的结果：",solution.simplifyPath(path))

例489 换硬币

3.代码实现

#参数coins是一个整数数组

#参数amount是硬币数的总金额

#返回值是可以换取的最少硬币数量

class Solution:

def coinChange(self, coins, amount):

import math

dp = [math.inf] \* (amount + 1)

dp[0] = 0

for i in range(amount + 1):

for j in range(len(coins)):

if i >= coins[j] and dp[i - coins[j]] < math.inf:

dp[i] = min(dp[i], dp[i - coins[j]] + 1)

if dp[amount] == math.inf:

return -1

else:

return dp[amount]

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

coins = [1, 2, 5]

amount = 11

print("硬币面额：", coins)

print("总硬币：", amount)

solution = Solution()

print("换取的最小硬币数量：", solution.coinChange(coins, amount))

例490 俄罗斯套娃信封

3.代码实现

#采用utf-8编码格式

#参数envelopes是一个整数对(w,h)，分别代表信封宽度和长度

#返回值是一个整数，代表最大的信封嵌套层数

class Solution:

def maxEnvelopes(self, envelopes):

height = [a[1] for a in sorted(envelopes, key = lambda x: (x[0], -x[1]))]

dp, length = [0] \* len(height), 0

import bisect

for h in height:

i = bisect.bisect\_left(dp, h, 0, length)

dp[i] = h

if i == length:

length += 1

return length

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

List = [[1,3],[8,5],[6,2]]

print(("输入："+str(List)))

print(("输出："+str(temp.maxEnvelopes(List))))

例491 木材加工

3.代码实现

#采用utf-8编码格式

#参数L是一个给定的n块木材的长度L[i]

#参数k是一个整数

#返回最小块的最大长度

class Solution:

def woodCut(self, L, k):

if not L:

return 0

start, end = 1, max(L)

while start + 1 < end:

mid = (start + end) // 2

if self.get\_pieces(L, mid) >= k:

start = mid

else:

end = mid

if self.get\_pieces(L, end) >= k:

return end

if self.get\_pieces(L, start) >= k:

return start

return 0

def get\_pieces(self, L, length):

pieces = 0

for l in L:

pieces += l // length

return pieces

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

L = [123,456,789]

k = 10

print("输入："+str(L))

print("输入："+str(k))

print("输出："+str(temp.woodCut(L,k)))

例492 判断数独是否合法

3.代码实现

#参数board是一个9行9列的二维数组  
#返回值是布尔值  
class Solution:  
 def isValidSudoku(self, board):  
 row = [set([]) for i in range(9)]  
 col = [set([]) for i in range(9)]  
 grid = [set([]) for i in range(9)]  
 for r in range(9):  
 for c in range(9):  
 if board[r][c] == '.':  
 continue  
 if board[r][c] in row[r]:  
 return False  
 if board[r][c] in col[c]:  
 return False  
 g = r // 3 \* 3 + c // 3  
 if board[r][c] in grid[g]:  
 return False  
 grid[g].add(board[r][c])  
 row[r].add(board[r][c])  
 col[c].add(board[r][c])  
 return True  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 board = [".87654321", "2........", "3........", "4........", "5........", "6........", "7........", "8........",  
 "9........"]  
 #创建对象  
 solution = Solution()  
 print("初始值：", board)  
 print("结果：", solution.isValidSudoku(board))

例493 斐波纳契数列

3.代码实现

#采用utf-8编码格式

class Solution:

def fibonacci(self, n):

a = 0

b = 1

for i in range(n - 1):

a, b = b, a + b

return a

if \_\_name\_\_ == '\_\_main\_\_':

temp = Solution()

nums1 = 5

nums2 = 15

print ("输入："+str(nums1))

print ("输出："+str(temp.fibonacci(nums1)))

print ("输入："+str(nums2))

print ("输出："+str(temp.fibonacci(nums2)))

例494 用栈模拟汉诺塔问题

3.代码实现

class Tower():  
 #创建三个汉诺塔，索引i从0～2  
 def \_\_init\_\_(self, i):  
 self.disks = []  
 #在汉诺塔上增加一个圆盘  
 def add(self, d):  
 if len(self.disks) > 0 and self.disks[-1] <= d:  
 print("Error placing disk %s" % d)  
 else:  
 self.disks.append(d)  
 #参数t是一个汉诺塔  
 #将塔最上面的一个圆盘移动到t的顶部  
 def move\_top\_to(self, t):  
 t.add(self.disks.pop())  
 #参数n是一个整数  
 #参数destination是一个汉诺塔  
 #参数buffer是一个汉诺塔  
 #将n个圆盘从此塔通过buffer塔移动到destination塔  
 def move\_disks(self, n, destination, buffer):  
 if n > 0:  
 self.move\_disks(n - 1, buffer, destination)  
 self.move\_top\_to(destination)  
 buffer.move\_disks(n - 1, destination, self)  
 def get\_disks(self):  
 return self.disks  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 towers = [Tower(0), Tower(1), Tower(2)]  
 n = 3  
 for i in range(n - 1, -1, -1):  
 towers[0].add(i)  
 towers[0].move\_disks(n, towers[2], towers[1])  
 print("初始盘子个数：", n)  
 print("towers[0]:", towers[0].disks, "towers[1]:", towers[1].disks, "towers[2]:", towers[2].disks)

例495 符号串生成器

3.代码实现

#参数generator是生成规则集合

#参数startSymbol是开始标志

#参数symbolString是标志字符串

#返回值是个布尔值，如果可以生成符号字符串则返回True，否则返回False

class Solution:

def getIdx(self, c):

return ord(c) - ord('A')

def nonTerminal(self, c):

return ord(c) >= ord('A') and ord(c) <= ord('Z')

def isMatched(self, s, pos, gen, sym):

if pos == len(s):

if len(gen) == 0:

return True

else:

return False

else:

if len(gen) == 0:

return False

elif self.nonTerminal(gen[0]):

idx = self.getIdx(gen[0])

for i in sym[idx]:

if self.isMatched(s, pos, i + gen[1:], sym):

return True

elif gen[0] == s[pos]:

if self.isMatched(s, pos + 1, gen[1:], sym):

return True

else:

return False

return False

def canBeGenerated(self, generator, startSymbol, symbolString):

sym = [[] for i in range(26)]

for i in generator:

sym[self.getIdx(i[0])].append(i[5:])

idx = self.getIdx(startSymbol)

for i in sym[idx]:

if self.isMatched(symbolString, 0, i, sym):

return True

return False

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

generator = ["S -> abc", "S -> aA", "A -> b", "A -> c"]

startSymbol = "S"

symbolString = "ac"

solution = Solution()

print("generator：", generator, ",startSymbol：", startSymbol, ",symbolString：", symbolString)

print("是否可以被生成", solution.canBeGenerated(generator, startSymbol, symbolString))

例496 移动零问题

3.代码实现

#参数nums是一个整数数组  
#返回排列后的数组  
class Solution:  
 def moveZeroes(self, nums):  
 left, right = 0, 0  
 while right < len(nums):  
 if nums[right] != 0:  
 if left != right:  
 nums[left] = nums[right]  
 left += 1  
 right += 1  
 while left < len(nums):  
 if nums[left] != 0:  
 nums[left] = 0  
 left += 1  
 return nums  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 nums = [0, 1, 0, 3, 12]  
 #创建对象  
 solution = Solution()  
 print("输入的整数数组 ：", nums)  
 nums=solution.moveZeroes(nums)  
 print("移动零后的数组:", nums)

例497 寻找数据错误

3.代码实现

#参数nums是一个数组  
#返回值是重复的数值和确实的数值  
class Solution:  
 def findErrorNums(self, nums):  
 n = len(nums)  
 hash = {}  
 result = []  
 sum = 0  
 for num in nums:  
 if num in hash:  
 result.append(num)  
 else:  
 hash[num] = 1  
 sum += num  
 result.append(int(n \* (n + 1) / 2) - sum)  
 return result  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 nums = [1, 2, 2, 4]  
 #创建对象  
 solution = Solution()  
 print("输入初始数组：", nums)  
 print("输出结果：", solution.findErrorNums(nums))

例498 找到映射序列

3.代码实现

#A的类型是整数数组  
#B的类型是整数数组  
#返回值的类型是整数数组  
class Solution:  
 def anagramMappings(self, A, B):  
 mapping = {v: k for k, v in enumerate(B)}  
 return [mapping[value] for value in A]  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 A = [12, 28, 46, 32, 50]  
 B = [50, 12, 32, 46, 28]  
 #创建对象  
 solution = Solution()  
 print("输入两个列表A= ", A, "B=", B)  
 print("输出结果：", solution.anagramMappings(A, B))

例499 旋转图像

3.代码实现

#参数matrix是整数数组的列表  
#没有返回值  
class Solution:  
 def rotate(self, matrix):  
 n = len(matrix)  
 for i in range(n):  
 for j in range(i + 1, n):  
 matrix[i][j], matrix[j][i] = matrix[j][i], matrix[i][j]  
 for i in range(n):  
 matrix[i].reverse()  
 return matrix  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 arr = [[1, 2], [3, 4]]  
 #创建对象  
 solution = Solution()  
 print("输入数组：", arr)  
 print("旋转后矩阵：", solution.rotate(arr))

例500 太平洋和大西洋的水流

3.代码实现

#参数matrix是给定的矩阵

#返回值是网格坐标列表

def inbound(x, y, n, m):

return 0 <= x < n and 0 <= y < m

class Solution:

def pacificAtlantic(self, matrix):

if not matrix or not matrix[0]:

return []

n, m = len(matrix), len(matrix[0])

p\_visited = [[False] \* m for \_ in range(n)]

a\_visited = [[False] \* m for \_ in range(n)]

for i in range(n):

self.dfs(matrix, i, 0, p\_visited)

self.dfs(matrix, i, m - 1, a\_visited)

for j in range(m):

self.dfs(matrix, 0, j, p\_visited)

self.dfs(matrix, n - 1, j, a\_visited)

res = []

for i in range(n):

for j in range(m):

if p\_visited[i][j] and a\_visited[i][j]:

res.append([i, j])

return res

def dfs(self, matrix, x, y, visited):

visited[x][y] = True

dx = [0, 1, 0, -1]

dy = [1, 0, -1, 0]

for i in range(4):

n\_x = dx[i] + x

n\_y = dy[i] + y

if not inbound(n\_x, n\_y, len(matrix), len(matrix[0])) or visited[n\_x][n\_y] or matrix[n\_x][n\_y] < matrix[x][

y]:

continue

self.dfs(matrix, n\_x, n\_y, visited)

#主函数

if \_\_name\_\_ == '\_\_main\_\_':

matrix = [[1, 2, 2, 3, 5], [3, 2, 3, 4, 4], [2, 4, 5, 3, 1], [6, 7, 1, 4, 5], [5, 1, 1, 2, 4]]

solution = Solution()

print("给定矩阵：", matrix)

print("满足条件的点坐标：", solution.pacificAtlantic(matrix))