### 【例1】完美平方

#### 3.代码实现

class Solution:

def numSquares(self, n):

while n % 4 == 0:

n //= 4

if n % 8 == 7:

return 4

for i in range(n + 1):

temp = i \* i

if temp <= n:

if int((n - temp) \*\* 0.5) \*\* 2 + temp == n:

return 1 + (0 if temp == 0 else 1)

else:

break

return 3

#主函数

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

n = 12

print("初始值：", n)

solution = Solution()

print("结果：", solution.numSquares(n))

### 【例2】判断平方数

#### 3.代码实现

class Solution:

def isPerfectSquare(self, num):

l = 0

r = num

while (r - l > 1):

mid = (l + r) / 2

if (mid \* mid <= num):

l = mid

else:

r = mid

ans = l

if (l \* l < num):

ans = r

return ans \* ans == num

#主函数

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

num = 16

print("初始值：", num)

solution = Solution()

print("结果：", solution.isPerfectSquare(num))

### 【例3】检测2的幂次

#### 3.代码实现

class Solution:

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()

nums1 = 16

nums2 = 17

print(("输入："+str(nums1)))

print(("输出："+str(temp.checkPowerOf2(nums1))))

print(("输入："+str(nums2)))

print(("输出："+str(temp.checkPowerOf2(nums2))))

### 【例4】求平方根

#### 3.代码实现

class Solution:

def sqrt(self, x):

l, r = 0, x

while l + 1 < r:

m = (r+l) // 2

if m \* m == x:

return m

elif m \* m > x:

r = m

else:

l = m

if l \* l == x:

return l

if r \* r == x:

return r

return l

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

temp = Solution()

x1 = 5

x2 = 10

print(("输入："+str(x1)))

print(("输出："+str(temp.sqrt(x1))))

print(("输入："+str(x2)))

print(("输出："+str(temp.sqrt(x2))))

### 【例5】x的n次幂

#### 3.代码实现

class Solution:

def myPow(self, x, n): #在Python3中整除需使用"//"

if n < 0 :

x = 1 // x

n = -n

ans = 1

tmp = x

while n != 0:

if n % 2 == 1:

ans \*= tmp

tmp \*= tmp

n //= 2

return ans

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

temp = Solution()

num1 = 123

num2 = 3

print(("输入："+str(num1)+" "+str(num2)))

print(("输出："+str(temp.myPow(num1,num2))))

### 【例6】快速幂

#### 3.代码实现

class Solution:

def fastPower(self, a, b, n):

ans = 1

while n > 0:

if n % 2 == 1:

ans = ans \* a % b

a = a \* a % b

n = n / 2

return ans % b

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

a = int(input("请输入a:"))

n = int(input("请输入n:"))

b = int(input("请输入b:"))

solution = Solution()

print("结果是：", solution.fastPower(a, n, b))

### 【例7】四数乘积

#### 3.代码实现

class Solution:

def numofplan(self, n, a, k):

sum = [0] \* 1000010

cnt = [0] \* 1000010

for i in range(n):

if a[i] > k:

continue

cnt[a[i]] += 1

for i in range(n):

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

if a[i] \* a[j] > k:

continue

sum[a[i] \* a[j]] += 1

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

cnt[i] += cnt[i - 1]

sum[i] += sum[i - 1]

ans = 0

for i in range(n):

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

res = a[i] \* a[j]

if res > k:

continue

res = k // res

ans += sum[res]

if a[i] <= res:

ans -= cnt[res // a[i]]

if a[i] <= res // a[i]:

ans += 1

if a[j] <= res:

ans -= cnt[res // a[j]]

if a[j] <= res // a[j]:

ans += 1

if a[i] \* a[j] <= res:

ans += 1

return ans // 6

#主函数

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

n = 5

a = [1, 1, 1, 2, 2]

k = 3

solution = Solution()

print("方案总数为：", solution.numofplan(n, a, k))

### 【例8】将整数A转换为B

#### 3.代码实现

class Solution:

def bitSwapRequired(self, a, b):

c = a ^ b

cnt = 0

for i in range(32):

if c & (1 << i) != 0:

cnt += 1

return cnt

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

temp = Solution()

a1 = 4; b1 = 45

a2 = 10; b2 = 26

print(("输入："+str(a1)+" "+str(b1)))

print(("输出："+str(temp.bitSwapRequired(a1,b1))))

print(("输入："+str(a2)+" "+str(b2)))

print(("输出："+str(temp.bitSwapRequired(a2,b2))))

### 【例9】罗马数字转整数

#### 3.代码实现

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10】整数转罗马数字

#### 3.代码实现

class Solution:

def parse(self, digit, index):

NUMS = {

1: 'I',

2: 'II',

3: 'III',

4: 'IV',

5: 'V',

6: 'VI',

7: 'VII',

8: 'VIII',

9: 'IX',

}

ROMAN = {

'I': ['I', 'X', 'C', 'M'],

'V': ['V', 'L', 'D', '?'],

'X': ['X', 'C', 'M', '?']

}

s = NUMS[digit]

return s.replace('X', ROMAN['X'][index]).replace('I', ROMAN['I'][index]).replace('V', ROMAN['V'][index])

def intToRoman(self, num):

s = ''

index = 0

while num != 0:

digit = num % 10

if digit != 0:

s = self.parse(digit, index) + s

num = num // 10

index += 1

return s

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

temp = Solution()

int1 = 56

int2 = 99

print(("输入："+str(int1)))

print(("输出："+str(temp.intToRoman(int1))))

print(("输入："+str(int2)))

print(("输出："+str(temp.intToRoman(int2))))

### 【例11】整数排序

#### 3.代码实现

class Solution:

def sortIntegers2(self, A):

self.quickSort(A, 0, len(A) - 1)

def quickSort(self, A, start, end):

if start >= end:

return

left, right = start, end

# key point 1: pivot is the value, not the index

pivot = A[int((start + end) / 2)]

# key point 2: every time you compare left & right, it should be

# left <= right not left < right

while left <= right:

while left <= right and A[left] < pivot:

left += 1

while left <= right and A[right] > pivot:

right -= 1

if left <= right:

A[left], A[right] = A[right], A[left]

left += 1

right -= 1

self.quickSort(A, start, right)

self.quickSort(A, left, end)

# 主函数

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

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

print('初始数组：', A)

solution = Solution()

solution.sortIntegers2(A)

print('快速排序:', A)

### 【例12】整数替换

#### 3.代码实现

class Solution:

def integerReplacement(self, n):

memo = {}

#if n == 1:

# return 0

self.dfs(n, memo)

print(memo[n])

return len(memo[n]) - 1

def dfs(self, n, memo):

temp = []

if n in memo:

return memo[n]

if n == 1:

temp.append(1)

memo[1] = temp

return temp

if n % 2 == 0:

temp.append(n)

cur = self.dfs(n // 2, memo)

temp.extend(cur)

memo[n] = temp

return temp

#temp.pop()

else:

temp2 = temp.copy()

n2 = n

temp.append(n)

cur = self.dfs((n + 1), memo)

temp.extend(cur)

temp2.append(n2)

cur2 = self.dfs((n2 - 1), memo)

temp2.extend(cur2)

if len(temp) < len(temp2):

memo[n] = temp

return temp

else:

memo[n] = temp2

return temp2

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

temp = Solution()

nums1 = 8

nums2 = 18

print(("输入："+str(nums1)))

print(("输出："+str(temp.integerReplacement(nums1))))

print(("输入："+str(nums2)))

print(("输出："+str(temp.integerReplacement(nums2))))

### 【例13】两个整数相除

#### 3.代码实现

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14】整数加法

#### 3.代码实现

class Solution:

def aplusb(self, a, b):

while b!=0:

a,b = (a^b)&0xffffffff,(a&b)<<1

return a

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

temp = Solution()

nums1 = 8

nums2 = 18

print(("输入："+str(nums1)+" "+str(nums2)))

print(("输出："+str(temp.aplusb(nums1,nums2))))

### 【例15】合并数字

#### 3.代码实现

import heapq

class Solution:

def mergeNumber(self, numbers):

Q = []

ans = 0

for i in numbers:

heapq.heappush(Q, i)

while(len(Q) > 1):

a = heapq.heappop(Q)

b = heapq.heappop(Q)

ans = ans + a + b

heapq.heappush(Q, a + b)

return ans

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

temp = Solution()

List1 = [1,2,3,4,5]

List2 = [6,7,8,9,10]

print(("输入："+str(List1)))

print(("输出："+str(temp.mergeNumber(List1))))

print(("输入："+str(List2)))

print(("输出："+str(temp.mergeNumber(List2))))

### 【例16】数字判断

#### 3.代码实现

class Solution:

def isNumber(self, s):

INVALID=0; SPACE=1; SIGN=2; DIGIT=3; DOT=4; EXPONENT=5;

#0是无效的，1空格，2符号，3数字，4小数点，5指数，6输入的数字

transitionTable=[[-1, 0, 3, 1, 2, -1], #状态0代表没有输入或者是空格

[-1, 8, -1, 1, 4, 5], #状态1输入是数字

[-1, -1, -1, 4, -1, -1], #状态2代表前面没有数字只有小数点

[-1, -1, -1, 1, 2, -1], #状态3代表符号

[-1, 8, -1, 4, -1, 5], #状态4代表数字其前方有小数点

[-1, -1, 6, 7, -1, -1], #状态5代表输入是'e'或者'E'

[-1, -1, -1, 7, -1, -1], #状态6代表在符号之后输入'e'

[-1, 8, -1, 7, -1, -1], #状态7代表在数字之后输入'e'

[-1, 8, -1, -1, -1, -1]] #状态8代表在输入有限输入后输入空格

state=0; i=0

while i<len(s):

inputtype = INVALID

if s[i]==' ': inputtype=SPACE

elif s[i]=='-' or s[i]=='+': inputtype=SIGN

elif s[i] in '0123456789': inputtype=DIGIT

elif s[i]=='.': inputtype=DOT

elif s[i]=='e' or s[i]=='E': inputtype=EXPONENT

state=transitionTable[state][inputtype]

if state==-1: return False

else: i+=1

return state == 1 or state == 4 or state == 7 or state == 8

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

temp = Solution()

string1 = "1"

string2 = "23"

print(("输入："+string1))

print(("输出："+str(temp.isNumber(string1))))

print(("输入："+string2))

print(("输出："+str(temp.isNumber(string2))))

### 【例17】下一个稀疏数

#### 3.代码实现

class Solution:

def nextSparseNum(self, x):

b\_x = bin(x)[2:]

pos = self.find\_highest\_continue\_one(b\_x)

while pos != -1:

if pos == 0:

b\_x = "1" + "0" \* len(b\_x)

else:

b\_x = b\_x[:pos - 1] + "1" + (len(b\_x) - pos) \* "0"

pos = self.find\_highest\_continue\_one(b\_x)

return int(b\_x, 2)

def find\_highest\_continue\_one(self, s):

n = len(s)

for i in range(n - 1):

if s[i] == s[i + 1] == "1":

return i

return -1

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

temp = Solution()

nums1 = 16

nums2 = 50

print(("输入："+str(nums1)))

print(("输出："+str(temp.nextSparseNum(nums1))))

print(("输入："+str(nums2)))

print(("输出："+str(temp.nextSparseNum(nums2))))

### 【例18】滑动窗口的最大值

#### 3.代码实现

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19】创建最大数

#### 3.代码实现

class Solution:

def maxNumber(self, nums1, nums2, k):

len1, len2 = len(nums1), len(nums2)

res = []

for x in range(max(0, k - len2), min(k, len1) + 1):

tmp = self.merge(self.getMax(nums1, x), self.getMax(nums2, k - x))

res = max(tmp, res)

return res

def getMax(self, nums, t):

ans = []

size = len(nums)

for x in range(size):

while ans and len(ans) + size - x > t and ans[-1] < nums[x]:

ans.pop()

if len(ans) < t:

ans.append(nums[x])

return ans

def merge(self, nums1, nums2):

return [max(nums1, nums2).pop(0) for \_ in nums1 + nums2]

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

temp = Solution()

List1 = [1,-1,-2,1]

List2 = [3,-2,2,1]

k = 3

print("输入："+str(List1))

print("输入："+str(List2))

print("输入："+str(k))

print(("输出："+str(temp.maxNumber(List1,List2,k))))

### 【例20】最接近的K个数

#### 3.代码实现

class Solution:

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

#找到 A[left] < target, A[right] >= target

#最接近target的两个数，肯定是相邻的

right = self.find\_upper\_closest(A, target)

left = right - 1

# 两根指针从中间往两边扩展，依次找到最接近k个数

results = []

for \_ in range(k):

if self.is\_left\_closer(A, target, left, right):

results.append(A[left])

left -= 1

else:

results.append(A[right])

right += 1

return results

def find\_upper\_closest(self, A, target):

# 找到A中第一个大于等于target的数字

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

while start + 1 < end:

mid = (start + end) // 2

if A[mid] >= target:

end = mid

else:

start = mid

if A[start] >= target:

return start

if A[end] >= target:

return end

# 找不到的情况

return end + 1

def is\_left\_closer(self, A, target, left, right):

if left < 0:

return False

if right >= len(A):

return True

return target - A[left] <= A[right] - target

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

temp = Solution()

A = [1,2,3]

target = 2

k = 3

print(("输出："+str(temp.kClosestNumbers(A,target,k))))

### 【例21】交错正负数

#### 3.代码实现

class Solution:

def subfun(self, A, B):

ans = []

for i in range(len(B)):

ans.append(A[i])

ans.append(B[i])

if(len(A) > len(B)):

ans.append(A[-1])

return ans

def rerange(self, A):

Ap = [i for i in A if i > 0]

Am = [i for i in A if i < 0]

if(len(Ap) > len(Am)):

tmp = self.subfun(Ap, Am)

else:

tmp = self.subfun(Am, Ap)

for i in range(len(tmp)):

A[i] = tmp[i];

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

temp = Solution()

List1 = [-1, -2, -3, 4, 5, 6]

List2 = [2,-4,6,8,-10]

print(("输入："+str(List1)))

temp.rerange(List1)

print(("输出："+str(List1)))

print(("输入："+str(List2)))

temp.rerange(List2)

print(("输出："+str(List2)))

### 【例22】下一个更大的数

#### 3.代码实现

class Solution:

def nextGreaterElements(self, nums):  
 if not nums:  
 return []  
 stack, res = [], [-1 for i in range(len(nums))]  
 for i in range(len(nums)):  
 if stack and nums[i] > nums[stack[-1]]:  
 while stack and nums[i] > nums[stack[-1]]:  
 pop\_index = stack.pop()  
 res[pop\_index] = nums[i]  
 stack.append(i)  
 for i in range(len(nums)):  
 if stack and nums[i] > nums[stack[-1]]:  
 while stack and nums[i] > nums[stack[-1]]:  
 pop\_index = stack.pop()  
 res[pop\_index] = nums[i]  
 stack.append(i)  
 if nums[stack[0]] == nums[stack[-1]]:  
 break  
 return res  
#主函数  
if \_\_name\_\_=="\_\_main\_\_":  
 nums=[1,2,1]  
 #创建对象  
 solution=Solution()  
 print("输入的数组是：",nums)  
 print("计算后的结果：",solution.nextGreaterElements(nums))

### 【例23】落单的数Ⅰ

#### 3.代码实现

class Solution:

def singleNumber(self, A):

ans = 0;

for x in A:

ans = ans ^ x

return ans

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

temp = Solution()

List1 = [4,6,4,6,3]

List2 = [2,1,1,1,1]

print(("输入："+str(List1)))

print(("输出："+str(temp.singleNumber(List1))))

print(("输入："+str(List2)))

print(("输出："+str(temp.singleNumber(List2))))

### 【例24】落单的数Ⅱ

#### 3.代码实现

class Solution:

def singleNumberII(self, A):

n = len(A)

d = [0 for i in range(32)]

for x in A:

for j in range(32):

if ( ((1 << j) & x) > 0):

d[j] += 1

ans = 0

for j in range(32):

t = d[j] % 3

if (t == 1):

ans = ans + (1 << j)

elif (t != 0):

return -1

return ans

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

temp = Solution()

List1 = [4,6,4,6,3,4,6]

List2 = [2,1,1,1,1,1,1]

print(("输入："+str(List1)))

print(("输出："+str(temp.singleNumberII(List1))))

print(("输入："+str(List2)))

print(("输出："+str(temp.singleNumberII(List2))))

### 【例25】落单的数Ⅲ

#### 3.代码实现

class Solution:

def singleNumberIII(self, A):

s = 0

for x in A:

s ^= x

y = s & (-s)

ans = [0,0]

for x in A:

if (x & y) != 0:

ans[0] ^= x

else:

ans[1] ^= x

return ans

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.singleNumberIII(List1))))

print(("输入："+str(str(List2))))

print(("输出："+str(temp.singleNumberIII(List2))))

### 【例26】落单的数Ⅳ

#### 3.代码实现

class Solution:

def getSingleNumber(self, nums):

left = 0

right = len(nums) - 1

while left < right:

mid = (left + right) // 2

if nums[mid] == nums[mid - 1]:

if (mid - left + 1) % 2 == 1:

right = mid - 2

else:

left = mid + 1

elif nums[mid] == nums[mid + 1]:

if (right - mid + 1) % 2 == 1:

left = mid + 2

else:

right = mid - 1

else:

return nums[mid]

return nums[left]

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

temp = Solution()

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

print(("输入："+str(nums)))

print(("输出："+str(temp.getSingleNumber(nums))))

### 【例27】对称数

#### 3.代码实现

import collections

class Solution:

def findStrobogrammatic(self, n):

ROTATE = {}

ROTATE["0"] = "0"

ROTATE["1"] = "1"

ROTATE["6"] = "9"

ROTATE["8"] = "8"

ROTATE["9"] = "6"

queue = collections.deque()

if n % 2 == 0:

queue.append("")

else:

queue.append("0")

queue.append("1")

queue.append("8")

result = []

while queue:

num = queue.popleft()

if len(num) == n:

result += [num] if num[0] != "0" or n == 1 else []

else:

for key, val in ROTATE.items():

queue.append(key + num + val)

return result

#主函数

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

n = 2

print("初始值：", n)

solution = Solution()

print("结果：", solution.findStrobogrammatic(n))

### 【例28】镜像数字

#### 3.代码实现

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("初始值是：", num)  
 print(" 结果是：", solution.isStrobogrammatic(num))

### 【例29】统计比给定整数小的数

#### 3.代码实现

class Solution:  
 def countOfSmallerNumber(self, A, queries):  
 A = sorted(A)  
 results = []  
 for q in queries:  
 results.append(self.countSmaller(A, q))  
 return results  
 def countSmaller(self, A, q):  
 # find the first number in A >= q

#找到A >= q的第一个数字  
 if len(A) == 0 or A[-1] < q:  
 return len(A)  
 start, end = 0, len(A) - 1  
 while start + 1 < end:  
 mid = (start + end) // 2  
 if A[mid] < q:  
 start = mid  
 else:  
 end = mid  
 if A[start] >= q:  
 return start  
 if A[end] >= q:  
 return end  
 return end + 1  
if \_\_name\_\_ == '\_\_main\_\_':  
 A = [1, 2, 7, 8, 5]  
 print("输入的数组是：", A)  
 solution = Solution()  
 print("数组中小于给定整数[1,8,5]的元素的数量是：", solution.countOfSmallerNumber(A, [1, 8, 5]))

### 【例30】统计前面比自己小的数

#### 3.代码实现

class SegTree:

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

self.start = start

self.end = end

self.left = None

self.right = None

self.count = 0

if start != end:

self.left = SegTree(start, (start + end) // 2)

self.right = SegTree((start + end) // 2 + 1, end)

def sum(self, start, end):

if start <= self.start and end >= self.end:

return self.count

if self.start == self.end:

return 0

if end <= self.left.end:

return self.left.sum(start, end)

if start >= self.right.start:

return self.right.sum(start, end)

return (self.left.sum(start, self.left.end) +

self.right.sum(self.right.start, end))

def inc(self, index):

if self.start == self.end:

self.count += 1

return

if index <= self.left.end:

self.left.inc(index)

else:

self.right.inc(index)

self.count = self.left.count + self.right.count

class Solution:

def countOfSmallerNumberII(self, A):

if len(A) == 0:

return []

root = SegTree(0, max(A))

results = []

for a in A:

results.append(root.sum(0, a - 1))

root.inc(a)

return results

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

temp = Solution()

nums = [6,4,7,2,3]

print(("输入："+str(nums)))

print(("输出："+str(temp.countOfSmallerNumberII(nums))))

### 【例31】阶乘尾部零的个数

#### 3.代码实现

class Solution:

def trailingZeros(self, n):

sum = 0

while n != 0:

n //= 5

sum += n

return sum

#主函数

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

n = 11

print("初始值：", n)

solution = Solution()

print("结果：", solution.trailingZeros(n))

### 【例32】统计数字

#### 3.代码实现

class Solution:

def digitCounts(self, k, n):

assert(n >= 0 and 0 <= k <= 9)

count = 0

for i in range(n + 1):

j = i

while True:

if j % 10 == k:

count += 1

j /= 10

if j == 0:

break

return count

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

temp = Solution()

k1 = 1

n1 = 11

k2 = 2

n2 = 22

print(("输入："+str(k1)+" "+str(n1)))

print(("输出："+str(temp.digitCounts(k1,n1))))

print(("输入："+str(k2)+" "+str(n2)))

print(("输出："+str(temp.digitCounts(k2,n2))))

### 【例33】删除数字

#### 3.代码实现

class Solution:

def DeleteDigits(self, A, k):

A = list(A)

while k > 0:

f = True

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

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

del A[i]

f = False

break

if f and len(A)>1:

A.pop()

k -= 1

while len(A)>1 and A[0]=='0':

del A[0]

return ''.join(A)

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

temp = Solution()

num\_str = "123456789"

k = 5

print(("输入："+num\_str+" "+str(k)))

print(("输出："+str(temp.DeleteDigits(num\_str,k))))

### 【例34】寻找丢失的数

#### 3.代码实现

class Solution:

def findMissing2(self, n, str):

used = [False for \_ in range(n + 1)]

return self.find(n, str, 0, used)

def find(self, n, str, index, used):

if index == len(str):

results = []

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

if not used[i]:

results.append(i)

return results[0] if len(results) == 1 else -1

if str[index] == '0':

return -1

for l in range(1, 3):

num = int(str[index: index + l])

if num >= 1 and num <= n and not used[num]:

used[num] = True

target = self.find(n, str, index + l, used)

if target != -1:

return target

used[num] = False

return -1

#主函数

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

n = 20

str = "19201234567891011121314151618"

print("n = ", n)

print("str = ", str)

solution = Solution()

print("缺少的数字是：", solution.findMissing2(n, str))

### 【例35】丑数Ⅰ

#### 3.代码实现

class Solution:

def isUgly(self, num):

if num <= 0:

return False

if num == 1:

return True

while num >= 2 and num % 2 == 0:

num /= 2;

while num >= 3 and num % 3 == 0:

num /= 3;

while num >= 5 and num % 5 == 0:

num /= 5;

return num == 1

#主函数

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

num = 8

print("初始值：", num)

solution = Solution()

print("是否是丑数：", solution.isUgly(num))

### 【例36】丑数Ⅱ

#### 3.代码实现

import heapq  
class Solution:  
 def nthUglyNumber(self, n):  
 heap = [1]  
 visited = set([1])  
 val = None  
 for i in range(n):  
 val = heapq.heappop(heap)  
 for multi in [2, 3, 5]:  
 if val \* multi not in visited:  
 visited.add(val \* multi)  
 heapq.heappush(heap, val \* multi)  
 return val  
if \_\_name\_\_ == '\_\_main\_\_':  
 n = 9  
 print("输入的n是：", n)  
 solution = Solution()  
 print("只含素因子2、3、5的第n小的数是：", solution.nthUglyNumber(n))

### 【例37】超级丑数

#### 3.代码实现

class Solution:

def nthSuperUglyNumber(self, n, primes):

import heapq

length = len(primes)

times = [0] \* length

uglys = [1]

minlist = [(primes[i] \* uglys[times[i]], i) for i in range(len(times))]

heapq.heapify(minlist)

while len(uglys) < n:

(umin, min\_times) = heapq.heappop(minlist)

times[min\_times] += 1

if umin != uglys[-1]:

uglys.append(umin)

heapq.heappush(minlist, (primes[min\_times] \* uglys[times[min\_times]], min\_times))

return uglys[-1]

#主函数

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

n = 6

primes = [2, 7, 13, 19]

print("初始值：", n)

print("质数集合：", primes)

solution = Solution()

print("第{}个丑数：".format(n), solution.nthSuperUglyNumber(n, primes))

### 【例38】两数之和Ⅰ

#### 3.代码实现

class Solution(object):

def twoSum(self, nums, target):

#hash用于建立数值到下标的映射

hash = {}

#循环nums数值，并添加映射

for i in range(len(nums)):

if target - nums[i] in hash:

return [hash[target - nums[i]], i]

hash[nums[i]] = i

#无解的情况

return [-1, -1]

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

temp = Solution()

List = [5,4,3,11]

nums = 5

print(("输入："+str(List)+" "+str(nums)))

print(("输出："+str(temp.twoSum(List,nums))))

### 【例39】两数之和Ⅱ

#### 3.代码实现

class Solution:  
 def twoSum(self, nums, target):  
 if not nums:  
 return []  
 left, right = 0, len(nums) - 1  
 while left < right:  
 res = target - nums[left]  
 if res == nums[right]:  
 break  
 elif res < nums[right]:  
 right -= 1  
 else:  
 left += 1  
 return [left + 1, right + 1]  
# 主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 nums = [2, 7, 11, 15]  
 target = 9  
 # 创建对象  
 solution = Solution()  
 print("初始化的数组nums=", nums, "目标值target=", target)  
 print(" 两个数的和等于目标值的下标是：", solution.twoSum(nums, target))

### 【例40】两数之和Ⅲ

#### 3.代码实现

class TwoSum:  
 data = []  
 def add(self, number):  
 self.data.append(number)  
 #参数value是一个整数  
 #返回值是找到存在的任意一对数字，使其和等于value值  
 def find(self, value):  
 self.data.sort()  
 left, right = 0, len(self.data) - 1  
 while left < right:  
 if self.data[left] + self.data[right] == value:  
 return True  
 if self.data[left] + self.data[right] < value:  
 left += 1  
 else:  
 right -= 1  
 return False  
# 主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 list = []  
 # 创建对象  
 solution = TwoSum()  
 solution.add(1)  
 solution.add(3)  
 solution.add(5)  
 list.append(solution.find(4))  
 list.append(solution.find(7))  
 print("初始化的输入顺序是add(1),add(2),add(3),find(4),find(7)")  
 print("输出的结果是：", list)

### 【例41】最接近的三数之和

#### 3.代码实现

class Solution:

def threeSumClosest(self, numbers, target):

numbers.sort()

ans = None

for i in range(len(numbers)):

left, right = i + 1, len(numbers) - 1

while left < right:

sum = numbers[left] + numbers[right] + numbers[i]

if ans is None or abs(sum - target) < abs(ans - target):

ans = sum

if sum <= target:

left += 1

else:

right -= 1

return ans

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

temp = Solution()

List1 = [1,2,3,4,5]

nums1 = 3

print(("输入："+str(List1)+" "+str(nums1)))

print(("输出："+str(temp.threeSumClosest(List1,nums1))))

### 【例42】三数之和为零

#### 3.代码实现

class Solution:

def threeSum(self, nums):

nums.sort()

results = []

length = len(nums)

for i in range(0, length - 2):

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

continue

target = -nums[i]

left, right = i + 1, length - 1

while left < right:

if nums[left] + nums[right] == target:

results.append([nums[i], nums[left], nums[right]])

right -= 1

left += 1

while left < right and nums[left] == nums[left - 1]:

left += 1

while left < right and nums[right] == nums[right + 1]:

right -= 1

elif nums[left] + nums[right] > target:

right -= 1

else:

left += 1

return results

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

temp = Solution()

List1 = [-1,-1,1,1,2,-2]

List2 = [3,0,2,-5,1]

print(("输入："+str(List1)))

print(("输出："+str(temp.threeSum(List1))))

print(("输入："+str(List2)))

print(("输出："+str(temp.threeSum(List2))))

### 【例43】四数之和为定值

#### 3.代码实现

class Solution(object):

def fourSum(self, nums, target):

nums.sort()

res = []

length = len(nums)

for i in range(0, length - 3):

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

continue

for j in range(i + 1, length - 2):

if j != i + 1 and nums[j] == nums[j - 1]:

continue

sum = target - nums[i] - nums[j]

left, right = j + 1, length - 1

while left < right:

if nums[left] + nums[right] == sum:

res.append([nums[i], nums[j], nums[left], nums[right]])

right -= 1

left += 1

while left < right and nums[left] == nums[left - 1]:

left += 1

while left < right and nums[right] == nums[right + 1]:

right -= 1

elif nums[left] + nums[right] > sum:

right -= 1

else:

left += 1

return res

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

temp = Solution()

List1 = [1,2,3,4,5,1]

nums1 = 10

print(("输入："+str(List1)+" "+str(nums1)))

print(("输出："+str(temp.fourSum(List1,nums1))))

### 【例44】骰子求和

#### 3.代码实现

class Solution:

def dicesSum(self, n):

results = []

f = [[0 for j in range(6 \* n + 1)] for i in range(n + 1)]

for i in range(1, 7):

f[1][i] = 1.0 / 6.0

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

for j in range(i, 6 \* n + 1):

for k in range(1, 7):

if j > k:

f[i][j] += f[i - 1][j - k]

f[i][j] /= 6.0

for i in range(n, 6 \* n + 1):

results.append((i, f[n][i]))

return results

# 主函数

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

n = 1

print("骰子的个数：", n)

solution = Solution()

print("结果：", solution.dicesSum(n))

### 【例45】k数之和

#### 3.代码实现

class Solution:

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

n = len(A)

dp = [

[[0] \* (target + 1) for \_ in range(k + 1)],

[[0] \* (target + 1) for \_ in range(k + 1)],

]

#dp[i][j][s]

#前i个数里挑出j个数，和为s

dp[0][0][0] = 1

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

dp[i % 2][0][0] = 1

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

for s in range(1, target + 1):

dp[i % 2][j][s] = dp[(i - 1) % 2][j][s]

if s >= A[i - 1]:

dp[i % 2][j][s] += dp[(i - 1) % 2][j - 1][s - A[i - 1]]

return dp[n % 2][k][target]

# 主函数

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

A = [1, 2, 3, 4]

k = 2

target = 5

print("初始数组A：", A)

print("整数k和目标值target：", k, target)

solution = Solution()

print("方案种类：", solution.kSum(A, k, target))

### 【例46】二进制求和

#### 3.代码实现

class Solution:

def addBinary(self, a, b):

indexa = len(a) - 1

indexb = len(b) - 1

carry = 0

sum = ""

while indexa >= 0 or indexb >= 0:

x = int(a[indexa]) if indexa >= 0 else 0

y = int(b[indexb]) if indexb >= 0 else 0

if (x + y + carry) % 2 == 0:

sum = '0' + sum

else:

sum = '1' + sum

carry = (x + y + carry) / 2

indexa, indexb = indexa - 1, indexb - 1

if carry == 1:

sum = '1' + sum

return sum

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

temp = Solution()

string1 = "1"

string2 = "10"

print(("输入："+string1+"+"+string2))

print(("输出："+str(temp.addBinary(string1,string2))))

### 【例47】各位相加

#### 3.代码实现

class Solution:

def addDigits(self, num):

self.num = list(str(num))

self.num = list(map(int, self.num))

self.num = sum(self.num)

if len(str(self.num)) == 1:

return self.num

elif len(str(self.num)) > 1:

self.addDigits(self.num)

return self.num

# 主函数

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

num = 38

print("初始值：", num)

solution = Solution()

print("结果：", solution.addDigits(num))

### 【例48】矩阵元素ZigZag返回

#### 3.代码实现

class Solution:

def printZMatrix(self, matrix):  
 if len(matrix) == 0:  
 return []  
 x, y = 0, 0  
 n, m = len(matrix), len(matrix[0])  
 rows, cols = range(n), range(m)  
 dx = [1, -1]  
 dy = [-1, 1]  
 direct = 1  
 result = []  
 for i in range(len(matrix) \* len(matrix[0])):  
 result.append(matrix[x][y])  
 nextX = x + dx[direct]  
 nextY = y + dy[direct]  
 if nextX not in rows or nextY not in cols:  
 if direct == 1:  
 if nextY >= m:  
 nextX, nextY = x + 1, y  
 else:  
 nextX, nextY = x, y + 1  
 else:  
 if nextX >= n:  
 nextX, nextY = x, y + 1  
 else:  
 nextX, nextY = x + 1, y  
 direct = 1 - direct  
 x, y = nextX, nextY  
 return result  
# 主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 matrim = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]  
 # 创建对象  
 solution = Solution()  
 print("输入的数组为：", matrim)  
 print("ZigZag顺序返回矩阵的所有元素是：", solution.printZMatrix(matrim))

### 【例49】子矩阵和为零

#### 3.代码实现

class Solution:  
 def submatrixSum(self, matrix):  
 if not matrix:  
 return []  
 wide = len(matrix[0])  
 depth = len(matrix)  
 res = None  
 prefixsum = [[0 for j in range(wide + 1)] for i in range(depth + 1)]  
 for dy in range(1, depth + 1):  
 for dx in range(1, wide + 1):  
prefixsum[dy][dx] = prefixsum[dy - 1][dx] + prefixsum[dy][dx - 1] - prefixsum[dy - 1][dx - 1] + \  
 matrix[dy - 1][dx - 1]  
 for y in range(dy):  
 for x in range(dx):  
 if prefixsum[dy][dx] == prefixsum[dy][x] + prefixsum[y][dx] - prefixsum[y][x]:  
 res = [(y, x), (dy - 1, dx - 1)]  
 return res  
# 主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 arr = [[1, 5, 7], [3, 7, -8], [4, -8, 9]]  
 # 创建对象  
 solution = Solution()  
 print("输入的数组为：", arr)  
 print("输出的结果是：", solution.submatrixSum(arr))

### 【例50】搜索二维矩阵Ⅰ

#### 3.代码实现

class Solution:  
 def searchMatrix(self, matrix, target):  
 if matrix == None or len(matrix) == 0:  
 return False  
 n, m = len(matrix), len(matrix[0])  
 x, y = 0, m - 1  
 while x <= n - 1 and y >= 0:  
 goal = matrix[x][y]  
 if target > goal:  
 x += 1  
 if target < goal:  
 y -= 1  
 if target == goal:  
 return True  
 return False  
# 主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 arr = [[1, 3, 5, 7], [10, 11, 16, 20], [23, 30, 34, 50]]  
 target = 3  
 # 创建对象  
 solution = Solution()  
 print("输入的整数数组是：", arr)  
 print("输入的目标值是", target)  
 print("输出的结果是：", solution.searchMatrix(arr, target))

### 【例51】搜索二维矩阵Ⅱ

#### 3.代码实现

class Solution:  
 def searchMatrix(self, matrix, target):  
 if matrix == [] or matrix[0] == []:  
 return 0  
 row, column = len(matrix), len(matrix[0])  
 i, j = row - 1, 0  
 count = 0  
 while i >= 0 and j < column:  
 if matrix[i][j] == target:  
 count += 1  
 i -= 1  
 j += 1  
 elif matrix[i][j] < target:  
 j += 1  
 elif matrix[i][j] > target:  
 i -= 1  
 return count  
# 主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 arr = [[1, 3, 5, 7], [2, 4, 7, 8], [3, 5, 9, 10]]  
 target = 3  
 # 创建对象  
 solution = Solution()  
 print("输入的数组是：", arr)  
 print("输入的目标值是：", target)  
 print("该目标出现的次数是：", solution.searchMatrix(arr, target))

### 【例52】矩阵归零

#### 3.代码实现

class Solution:  
 def setZeroes(self, matrix):  
 if len(matrix) == 0:  
 return  
 rownum = len(matrix)  
 colnum = len(matrix[0])  
 row = [False for i in range(rownum)]  
 col = [False for i in range(colnum)]  
 for i in range(rownum):  
 for j in range(colnum):  
 if matrix[i][j] == 0:  
 row[i] = True  
 col[j] = True  
 for i in range(rownum):  
 for j in range(colnum):  
 if row[i] or col[j]:  
 matrix[i][j] = 0  
 return matrix  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 arr = [[1, 2], [0, 3]]  
 #创建对象  
 solution = Solution()  
 print("输入的数组是：", arr)  
 print("变换后的矩阵是：", solution.setZeroes(arr))

### 【例53】和为零的子矩阵

#### 3.代码实现

class Solution:

def submatrixSum(self, matrix):

if not matrix or not matrix[0]:

return None

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

for top in range(n):

arr = [0] \* m

for down in range(top, n):

prefix\_hash = {0: -1}

prefix\_sum = 0

for col in range(m):

arr[col] += matrix[down][col]

prefix\_sum += arr[col]

if prefix\_sum in prefix\_hash:

return [(top, prefix\_hash[prefix\_sum] + 1), (down, col)]

prefix\_hash[prefix\_sum] = col

return None

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

temp = Solution()

nums1 = [[1,5,7],[3,7,-8],[4,-8,9]]

nums2 = [[2,1,4],[-6,7,4],[-4,3,-5]]

print ("输入矩阵："+str(nums1))

print ("输出："+str(temp.submatrixSum(nums1)))

print ("输入矩阵："+str(nums2))

print ("输出："+str(temp.submatrixSum(nums2)))

### 【例54】螺旋矩阵

#### 3.代码实现

#参数matrix是mxn的矩阵  
#返回值是一个整数数组  
class Solution:

def spiralOrder(self, matrix):  
 if matrix == []: return []  
 up = 0;  
 left = 0  
 down = len(matrix) - 1  
 right = len(matrix[0]) - 1  
 direct = 0 # 0: 向右 1:向下 2:向左 3:向上  
 res = []  
 while True:  
 if direct == 0:  
 for i in range(left, right + 1):  
 res.append(matrix[up][i])  
 up += 1  
 if direct == 1:  
 for i in range(up, down + 1):  
 res.append(matrix[i][right])  
 right -= 1  
 if direct == 2:  
 for i in range(right, left - 1, -1):  
 res.append(matrix[down][i])  
 down -= 1  
 if direct == 3:  
 for i in range(down, up - 1, -1):  
 res.append(matrix[i][left])  
 left += 1  
 if up > down or left > right: return res  
 direct = (direct + 1) % 4  
#主函数  
if \_\_name\_\_=="\_\_main\_\_":  
 arr = [[ 1, 2, 3 ],[ 4, 5, 6 ],[ 7, 8, 9 ]]  
 #创建对象  
 solution=Solution()  
 print("输入的数组是：", arr)  
 print("螺旋顺序顺出后的矩阵是：",solution.spiralOrder(arr))

### 【例55】矩阵走路问题

#### 3.代码实现

class node:

def \_\_init\_\_(self, a=0, b=0, i=0, s=0):

self.x = a

self.y = b

self.i = i

self.step = s

class Solution:

def getBestRoad(self, grid):

direction = [[1, 0], [-1, 0], [0, 1], [0, -1]]

n = len(grid)

m = len(grid[0])

# print(n,m)

visit = [[[0 for i in range(2)] for i in range(m)] for i in range(n)]

p = []

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

new = node(0, 0, 0, 0)

visit[0][0][0] = 1;

else:

new = node(0, 0, 1, 0)

p.append(new)

flag = -1;

visit[0][0][1] = 1;

cnt = 0

while cnt < len(p):

a = p[cnt]

cnt += 1

# print(a.x,a.y,a.i,a.step)

if a.x == n - 1 and a.y == m - 1:

flag = a.step

break

else:

for i in range(0, 4):

new\_x = a.x + direction[i][0]

new\_y = a.y + direction[i][1]

if new\_x <= n - 1 and new\_x >= 0 and new\_y <= m - 1 and new\_y >= 0:

if grid[new\_x][new\_y] == 0 and visit[new\_x][new\_y][a.i] == 0:

visit[new\_x][new\_y][a.i] = 1

visit[new\_x][new\_y][1] = 1

p.append(node(new\_x, new\_y, a.i, a.step + 1))

if grid[new\_x][new\_y] == 1 and a.i == 0 and visit[new\_x][new\_y][1] == 0:

visit[new\_x][new\_y][1] = 1

p.append(node(new\_x, new\_y, 1, a.step + 1))

return flag

# 主函数

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

a = [[0, 1, 0, 0, 0], [0, 0, 0, 1, 0], [1, 1, 0, 1, 0], [1, 1, 1, 1, 0]]

print("地图是：", a)

solution = Solution()

print("最少要走：", solution.getBestRoad(a))

### 【例56】稀疏矩阵乘法

#### 3.代码实现

class Solution:  
 def multiply(self, A, B):  
 n = len(A)  
 m = len(A[0])  
 k = len(B[0])  
 C = [[0 for \_ in range(k)] for i in range(n)]  
 for i in range(n):  
 for j in range(m):  
 if A[i][j] != 0:  
 for l in range(k):  
 C[i][l] += A[i][j] \* B[j][l]  
 return C  
# 主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 A = [[1, 0, 0], [-1, 0, 3]]  
 B = [[7, 0, 0], [0, 0, 0], [0, 0, 1]]  
 # 创建对象  
 solution = Solution()  
 print("输入的两个数组是A=", A, "B=", B)  
 print("输出的结果是：", solution.multiply(A, B))

### 【例57】直方图最大矩形覆盖

#### 3.代码实现

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("输入每个直方图的高度：",heights)  
 print("找到的直方图的最大面积：",solution.largestRectangleArea(heights))

### 【例58】最大矩形

#### 3.代码实现

class Solution:

def maximalRectangle(self, matrix):  
 if not matrix:  
 return 0  
 max\_rectangle = 0  
 heights = [0] \* len(matrix[0])  
 for row in matrix:  
 for index, num in enumerate(row):  
 heights[index] = heights[index] + 1 if num else 0  
 max\_rectangle = max(  
 max\_rectangle,  
 self.find\_max\_rectangle(heights),  
 )  
 return max\_rectangle  
 def find\_max\_rectangle(self, heights):  
 indices\_stack = []  
 max\_rectangle = 0  
 for index, height in enumerate(heights + [-1]):  
 while indices\_stack and heights[indices\_stack[-1]] >= height:  
 popped = indices\_stack.pop(-1)  
 left\_bound = indices\_stack[-1] if indices\_stack else -1  
 max\_rectangle = max(  
 max\_rectangle,  
 (index - left\_bound - 1) \* heights[popped],  
 )  
 indices\_stack.append(index)  
 # print(indices\_stack)  
 return max\_rectangle  
#主函数  
if \_\_name\_\_=="\_\_main\_\_":  
 matrix=[[1,1,0,0,1],[0,1,0,0,1],[0,0,1,1,1],[0,0,1,1,1],[0,0,0,0,1]]  
 #创建对象  
 solution=Solution()  
 print("输入的布尔类型的二维矩阵是：",matrix)  
 print("最大的矩阵的面积是：",solution.maximalRectangle(matrix))

### 【例59】排序矩阵中的从小到大第k个数

#### 3.代码实现

class Solution:  
 def kthSmallest(self, matrix, k):  
 if not matrix or not matrix[0] or k == 0:  
 return None  
 while len(matrix) > 1:  
 matrix.append(self.merge(matrix.pop(0), matrix.pop(0)))  
 return matrix[0][k - 1]  
 def merge(self, nums1, nums2):  
 res, index1, index2 = [], 0, 0  
 while index1 < len(nums1) or index2 < len(nums2):  
 if index1 >= len(nums1):  
 res.append(nums2[index2])  
 index2 += 1  
 elif index2 >= len(nums2):  
 res.append(nums1[index1])  
 index1 += 1  
 elif nums1[index1] < nums2[index2]:  
 res.append(nums1[index1])  
 index1 += 1  
 else:  
 res.append(nums2[index2])  
 index2 += 1  
 return res  
# 创建主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 arr = [[1, 5, 7], [3, 7, 8], [4, 8, 9]]  
 index = 4  
 # 创建对象  
 solution = Solution()  
 print("输入的数组是：", arr)  
 print("运行后的结果是:", solution.kthSmallest(arr, index))

### 【例60】最大和子数组

#### 3.代码实现

import sys

class Solution:

def maxSubArray(self, nums):

min\_sum, max\_sum = 0, -sys.maxsize

prefix\_sum = 0

for num in nums:

prefix\_sum += num

max\_sum = max(max\_sum, prefix\_sum - min\_sum)

min\_sum = min(min\_sum, prefix\_sum)

return max\_sum

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

temp = Solution()

nums1 = [-1,-2,3,4,2,2,4,3,-6]

nums2 = [4,2,1,4,-1,2,7,4,-3]

print ("输入的数组："+"[-1,-2,3,4,2,2,4,3,-6]")

print ("输出："+str(temp.maxSubArray(nums1)))

print ("输入的数组："+'[4,2,1,4,-1,2,7,4,-3]')

print ("输出："+str(temp.maxSubArray(nums2)))

### 【例61】两个不重叠子数组最大和

#### 3.代码实现

class Solution:

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()

nums1 = [6,5,4,3,2]

nums2 = [2,1,2,1,2,1]

print(("输入："+str(nums1)))

print(("输出："+str(temp.maxTwoSubArrays(nums1))))

print(("输入："+str(nums2)))

print(("输出："+str(temp.maxTwoSubArrays(nums2))))

### 【例62】k个不重叠子数组最大和

#### 3.代码实现

class Solution:

def maxSubArray(self, nums, k):

#

MIN = - 2 \*\* 32

n = len(nums)

array = [0]

for num in nums:

array.append(num)

# include the last num

ans1 = [[MIN for i in range(k + 1)] for j in range(n + 1)]

# do not include the last num

ans2 = [[MIN for i in range(k + 1)] for j in range(n + 1)]

for i in range(n + 1):

ans1[i][0] = 0

ans2[i][0] = 0

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

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

ans1[i][j] = max(ans1[i - 1][j] + array[i], ans1[i - 1][j - 1] + array[i],

ans2[i - 1][j - 1] + array[i])

ans2[i][j] = max(ans1[i - 1][j], ans2[i - 1][j])

return max(ans1[n][k], ans2[n][k])

# 主函数

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

nums = [-1, 4, -2, 3, -2, 3]

k = 2

print("初始数组和k值：", nums, k)

solution = Solution()

print("不重叠子数组的和：", solution.maxSubArray(nums, k))

#### 3.代码实现

class Solution:

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()

nums1 = [5,3,1,-4]

nums2 = [3,-1,6,2]

print ("输入数组："+str(nums1))

print ("输出："+str(temp.maxDiffSubArrays(nums1)))

print ("输入数组："+str(nums2))

print ("输出："+str(temp.maxDiffSubArrays(nums2)))

### 【例64】两数组的交集Ⅰ

#### 3.代码实现

class Solution:

def intersection(self, nums1, nums2):

return list(set(nums1) & set(nums2))

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

temp = Solution()

List1 = [1,2,3,4]

List2 = [2,4,6,8]

print("输入："+str(List1)+" "+str(List2))

print(("输出："+str(temp.intersection(List1,List2))))

### 数组的交集Ⅱ

#### 3.代码实现

import collections

class Solution:

def intersection(self, nums1, nums2):

counts = collections.Counter(nums1)

result = []

for num in nums2:

if counts[num] > 0:

result.append(num)

counts[num] -= 1

return result

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

temp = Solution()

List1 = [1,2,3,4,5,6]

List2 = [2,4,6,8,10]

print(("输入："+str(List1)+" "+str(List2)))

print(("输出："+str(temp.intersection(List1,List2))))

### 【例66】乘积小于k的子数组

#### 3.代码实现

from collections import deque

class Solution:

def numSubarrayProductLessThanK(self, nums, k):

if not nums:

return 0

ans, product, index = 0, 1, 0

queue = deque()

while index < len(nums):

product \*= nums[index]

queue.append(nums[index])

while product >= k and queue:

remove = queue.popleft()

product /= remove

if queue:

ans += len(queue)

index += 1

return ans

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

temp = Solution()

List1 = [8,4,3,6,10]

num = 100

print("输入："+str(List1)+" "+str(num))

print(("输出："+str(temp.numSubarrayProductLessThanK(List1,num))))

### 【例67】最小和子数组

#### 3.代码实现

class Solution:

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()

List1 = [1,-1,-2,1]

List2 = [3,-2,2,1]

print("输入："+str(List1))

print(("输出："+str(temp.minSubArray(List1))))

print("输入："+str(List2))

print(("输出："+str(temp.minSubArray(List2))))

### 【例68】连续子数组最大和

#### 3.代码实现

class Solution:  
 def continuousSubarraySum(self, A):  
 ans = -0x7fffffff  
 sum = 0  
 start, end = 0, -1  
 result = [-1, -1]  
 for x in A:  
 if sum < 0:  
 sum = x  
 start = end + 1  
 end = start  
 else:  
 sum += x  
 end += 1  
 if sum > ans:  
 ans = sum  
 result = [start, end]  
 return result  
# 主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 nums=[-3, 1, 3, -3, 4]  
 # 创建对象  
 solution = Solution()  
 print("输入的数组是 ：", nums)  
 print("使得和最大的子数组是:",solution.continuousSubarraySum(nums))

### 【例69】子数组之和为零

#### 3.代码实现

class Solution:  
 def subarraySum(self, nums):  
 prefix\_hash = {0: -1}  
 prefix\_sum = 0  
 for i, num in enumerate(nums):  
 prefix\_sum += num  
 if prefix\_sum in prefix\_hash:  
 return prefix\_hash[prefix\_sum] + 1, i  
 prefix\_hash[prefix\_sum] = i  
 return -1, -1  
# 主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 nums = [-3, 1, 2, -3, 4]  
 # 创建对象  
 solution = Solution()  
 print("初始化的数组是：", nums)  
 print("和为零的子数组是：", solution.subarraySum(nums))

### 【例70】数组划分

#### 3.代码实现

class Solution:

def partitionArray(self, nums, k):

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

while start <= end:

while start <= end and nums[start] < k:

start += 1

while start <= end and nums[end] >= k:

end -= 1

if start <= end:

nums[start], nums[end] = nums[end], nums[start]

start += 1

end -= 1

return start

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

temp = Solution()

List1 = [5,1,4,2,3]

num = 2

print(("输入："+str(List1)+" "+str(num)))

print(("输出："+str(temp.partitionArray(List1,num))))

### 【例71】数组中的K-diff对的数量

#### 3.代码实现

class Solution:

def findPairs(self, nums, k):

if k > 0:

return len(set(nums) & set(n + k for n in nums))

if k == 0: #计数所有出现的数字> 1

return sum(v > 1 for v in collections.Counter(nums).values())

return 0

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

temp = Solution()

List1 = [6,3,4,2,5,1]

num = 2

print(("输入："+str(List1)+" "+str(num)))

print(("输出："+str(temp.findPairs(List1,num))))

### 【例72】删除排序数组中的重复数字

#### 3.代码实现

class Solution:

def removeDuplicates(self, A):

if A == []:

return 0

index = 0

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

if A[index] != A[i]:

index += 1

A[index] = A[i]

return index + 1

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

temp = Solution()

List1 = [1,2,3]

List2 = [2,5,1,3]

print(("输入："+str(List1)))

print(("输出："+str(temp.removeDuplicates(List1))))

print(("输入："+str(List2)))

print(("输出："+str(temp.removeDuplicates(List2))))

### 【例73】和大于定值的最小长度子数组

#### 3.代码实现

class Solution:

def minimumSize(self, nums, s):

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

return -1

n = len(nums)

minLength = n + 1

sum = 0

j = 0

for i in range(n):

while j < n and sum < s:

sum += nums[j]

j += 1

if sum >= s:

minLength = min(minLength, j - i)

sum -= nums[i]

if minLength == n + 1:

return -1

return minLength

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

temp = Solution()

List1 = [1,2,3,4,5]

nums1 = 10

print(("输入："+str(List1)+" "+str(nums1)))

print(("输出："+str(temp.minimumSize(List1,nums1))))

### 【例74】最大平均值子数组

#### 3.代码实现

class Solution:

def maxAverage(self, nums, k):

if not nums:

return 0

start, end = min(nums), max(nums)

while end - start > 1e-5:

mid = (start + end) / 2

if self.check\_subarray(nums, k, mid):

start = mid

else:

end = mid

return start

def check\_subarray(self, nums, k, average):

prefix\_sum = [0]

for num in nums:

prefix\_sum.append(prefix\_sum[-1] + num - average)

min\_prefix\_sum = 0

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

if prefix\_sum[i] - min\_prefix\_sum >= 0:

return True

min\_prefix\_sum = min(min\_prefix\_sum, prefix\_sum[i - k + 1])

return False

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

temp = Solution()

nums = [5,3,-4,6,-7,2,-1]

k = 5

print(("输入："+str(nums)+" "+str(k)))

print(("输出："+str(temp.maxAverage(nums,k))))

### 【例75】搜索旋转排序数组中的最小值Ⅰ

#### 3.代码实现

class Solution:

def findMin(self, nums):

if not nums:

return -1

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

target = nums[-1]

while start + 1 < end:

mid = (start + end) // 2

if nums[mid] <= target:

end = mid

else:

start = mid

return min(nums[start], nums[end])

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

temp = Solution()

List1 = [1,2,3,4,5]

List2 = [6,7,8,9,10]

print(("输入："+str(List1)))

print(("输出："+str(temp.findMin(List1))))

print(("输入："+str(List2)))

print(("输出："+str(temp.findMin(List2))))

### 【例76】搜索旋转排序数组中的最小值II

#### 3.代码实现

class Solution:

def findMin(self, num):

min = num[0]

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

while start<end:

mid = (start+end)//2

if num[mid]>num[end]:

start = mid+1

elif num[mid]<num[end]:

end = mid

else:

end = end - 1

return num[start]

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

temp = Solution()

List1 = [1,2,4,5,6,7,8]

print(("输入："+str(List1)))

print(("输出："+str(temp.findMin(List1))))

### 【例77】搜索旋转排序数组目标值Ⅰ

#### 3.代码实现

class Solution:

def search(self, A, target):

if not A:

return -1

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

while start + 1 < end:

mid = (start + end) // 2

if A[mid] >= A[start]:

if A[start] <= target <= A[mid]:

end = mid

else:

start = mid

else:

if A[mid] <= target <= A[end]:

start = mid

else:

end = mid

if A[start] == target:

return start

if A[end] == target:

return end

return -1

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

temp = Solution()

List1 = [1,2,3,4,5]; k1 = 5

List2 = [6,7,8,9,10]; k2 = 8

print(("输入："+str(List1)+" "+str(k1)))

print(("输出："+str(temp.search(List1,k1))))

print(("输入："+str(List2)+" "+str(k2)))

print(("输出："+str(temp.search(List2,k2))))

### 【例78】搜索旋转排序数组目标值Ⅱ

#### 3.代码实现

class Solution:

def search(self, A, target):

for num in A:

if num == target:

return True

return False

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

temp = Solution()

List1 = [1,2,4,5,6,7,8]

target = 5

print(("输入："+str(List1)+" "+str(target)))

print(("输出："+str(temp.search(List1,target))))

### 【例79】最接近零的子数组和

#### 3.代码实现

import sys

class Solution:

def subarraySumClosest(self, nums):

prefix\_sum = [(0, -1)]

for i, num in enumerate(nums):

prefix\_sum.append((prefix\_sum[-1][0] + num, i))

prefix\_sum.sort()

closest, answer = sys.maxsize, []

for i in range(1, len(prefix\_sum)):

if closest > prefix\_sum[i][0] - prefix\_sum[i - 1][0]:

closest = prefix\_sum[i][0] - prefix\_sum[i - 1][0]

left = min(prefix\_sum[i - 1][1], prefix\_sum[i][1]) + 1

right = max(prefix\_sum[i - 1][1], prefix\_sum[i][1])

answer = [left, right]

return answer

# 主函数

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

nums = [-3, 1, 1, -3, 5]

print("初始数组：", nums)

solution = Solution()

print("结果：", solution.subarraySumClosest(nums))

### 【例80】两个整数数组的最小差

#### 3.代码实现

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

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

temp = Solution()

List1 = [5,6,4,2,3]

List2 = [1,1,1,1,1]

print(("输入："+str(List1)+" "+str(List2)))

print(("输出："+str(temp.smallestDifference(List1,List2))))

### 【例81】数组中的相同数字

#### 3.代码实现

class Solution:  
 def sameNumber(self, nums, k):  
 vis = {}  
 n = len(nums)  
 for i in range(n):  
 x = nums[i]  
 if x in vis:  
 if i - vis[x] < k:  
 return "YES"  
 vis[x] = i  
 return "NO"  
# 主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 nums=[1,2,3,1,5,9,3]  
 k=4  
 #创建对象  
 solution=Solution()  
 print("输入的数是：",nums,"给定的k=",k)  
 print("输出的结果是：",solution.sameNumber(nums,k))

### 【例82】翻转数组

#### 3.代码实现

class Solution:  
 def reverseArray(self, nums):  
 start = 0  
 end = -1  
 for \_ in range(len(nums)//2):  
 nums[start], nums[end] = nums[end], nums[start]  
 start += 1  
 end -= 1  
 return nums  
#主函数  
if \_\_name\_\_=="\_\_main\_\_":  
 nums=[1,2,5]  
 #创建对象  
 solution=Solution()  
 print("输入的数组是：",nums)  
 print("翻转之后的结果是：",solution.reverseArray(nums))

### 【例83】奇偶分割数组

#### 3.代码实现

class Solution:  
 def partitionArray(self, nums):  
 start, end = 0, len(nums)-1  
 while start<end:  
 while start<end and nums[start]%2==1: start += 1  
 while start<end and nums[end]%2==0: end -=1  
 if start<end:  
 nums[start], nums[end] = nums[end], nums[start]  
 start += 1  
 end -= 1  
 return nums  
if \_\_name\_\_=="\_\_main\_\_":  
 nums = [1, 2, 3, 4]  
 solution=Solution()  
 print("输入的数组是：",nums)  
 print("奇偶分割数组后的结果是：",solution.partitionArray(nums))

### 【例84】判断字符串中的重复字符

#### 3.代码实现

class Solution:  
 def isUnique(self, str):  
 if len(str) > len(set(str)):  
 return False  
 else:  
 return True  
if \_\_name\_\_ == "\_\_main\_\_":  
 str = "abc"  
 solution = Solution()  
 print("输入的字符串是：", str)  
 print("输出的结果是：", solution.isUnique(str))

### 【例85】最长无重复字符的子串

#### 3.代码实现

class Solution:

def lengthOfLongestSubstring(self, s):

unique\_chars = set([])

j = 0

n = len(s)

longest = 0

for i in range(n):

while j < n and s[j] not in unique\_chars:

unique\_chars.add(s[j])

j += 1

longest = max(longest, j - i)

unique\_chars.remove(s[i])

return longest

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

temp = Solution()

string1 = "abccd"

string2 = "hahah"

print(("输入："+string1))

print(("输出："+str(temp.lengthOfLongestSubstring(string1))))

print(("输入："+string2))

print(("输出："+str(temp.lengthOfLongestSubstring(string2))))

### 【例86】最长回文子串

#### 3.代码实现

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87】转换字符串到整数

#### 3.代码实现

import sys

class Solution(object):

def atoi(self, str):

str = str.strip()

if str == "" :

return 0

i = 0

sign = 1

ret = 0

length = len(str)

MaxInt = (1 << 31) - 1

if str[i] == '+':

i += 1

elif str[i] == '-' :

i += 1

sign = -1

for i in range(i, length) :

if str[i] < '0' or str[i] > '9' :

break

ret = ret \* 10 + int(str[i])

if ret > sys.maxsize:

break

ret \*= sign

if ret >= MaxInt:

return MaxInt

if ret < MaxInt \* -1 :

return MaxInt \* - 1 - 1

return ret

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

temp = Solution()

string1 = "150"

string2 = "32"

print(("输入："+string1))

print(("输出："+str(temp.atoi(string1))))

print(("输入："+string2))

print(("输出："+str(temp.atoi(string2))))

### 【例88】字符串查找

#### 3.代码实现

class Solution:

def strStr(self, source, target):

if source is None or target is None:

return -1

len\_s = len(source)

len\_t = len(target)

for i in range(len\_s - len\_t + 1):

j = 0

while (j < len\_t):

if source[i + j] != target[j]:

break

j += 1

if j == len\_t:

return i

return -1

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

temp = Solution()

string1 = "abcd"

string2 = "cd"

print(("输入："+string1+" "+string2))

print(("输出："+str(temp.strStr(string1,string2))))

### 【例89】子字符串的判断

#### 3.代码实现

import collections

class Solution:

def checkInclusion(self, s1, s2):

len1, len2 = len(s1), len(s2)

if not s2 or len2 < len1:

return False

if not s1:

return True

window\_diff = collections.defaultdict(int)

for c in s1:

window\_diff[c] -= 1

for i in range(len1):

char = s2[i]

window\_diff[char] += 1

if window\_diff[char] == 0:

window\_diff.pop(char)

if len(window\_diff) == 0:

return True

for i in range(len1, len2):

char = s2[i]

char2rm = s2[i-len1]

window\_diff[char] += 1

window\_diff[char2rm] -= 1

if window\_diff[char] == 0:

window\_diff.pop(char)

if window\_diff[char2rm] == 0:

window\_diff.pop(char2rm)

if len(window\_diff) == 0:

return True

return False

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

temp = Solution()

string1 = "abc"

string2 = "dauwkbfyacb"

print("输入："+string1+" "+string2)

print(("输出："+str(temp.checkInclusion(string1,string2))))

### 【例90】翻转字符串中的单词

#### 3.代码实现

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91】乱序字符串

#### 3.代码实现

class Solution:

def anagrams(self, strs):

dict = {}

for word in strs:

sortedword = ''.join(sorted(word))

dict[sortedword] = [word] if sortedword not in dict 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92】比较字符串

#### 3.代码实现

class Solution:  
 def missingString(self, str1, str2):  
 result = []  
 dict = set(str2.split())  
 for word in str1.split():  
 if word not in dict:  
 result.append(word)  
 return result  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 str1 = "This is an example"  
 str2 = "is example"  
  
 solution = Solution()  
 print("输入的两个字符串是str1=", str1, "str2=", str2)  
 print("输出的结果是：", solution.missingString(str1, str2))

### 【例93】攀爬字符串

#### 3.代码实现

class Solution:

def isScramble(self, s1, s2):

if len(s1) != len(s2): return False

if s1 == s2: return True

l1 = list(s1);

l2 = list(s2)

l1.sort();

l2.sort()

if l1 != l2: return False

length = len(s1)

for i in range(1, length):

if self.isScramble(s1[:i], s2[:i]) and self.isScramble(s1[i:], s2[i:]): return True

if self.isScramble(s1[:i], s2[length - i:]) and self.isScramble(s1[i:], s2[:length - i]): return True

return False

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

s1, s2 = "great", "rgeat"

print("字符串s1：", s1)

print("字符串s2：", s2)

solution = Solution()

print("s2 是否为 s1 的攀爬字符串：", solution.isScramble(s1, s2))

### 【例94】交叉字符串

#### 3.代码实现

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)]

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95】字符串解码

#### 3.代码实现

class Solution:  
 def expressionExpand(self, s):  
 stack = []  
 for c in s:  
 if c != ']':  
 stack.append(c)  
 continue  
 strs = []  
 while stack and stack[-1] != '[':  
 strs.append(stack.pop())  
 # skip '['，跳过'['  
 stack.pop()  
 repeats = 0  
 base = 1  
 while stack and stack[-1].isdigit():  
 repeats += (ord(stack.pop()) - ord('0')) \* base  
 base \*= 10  
 stack.append(''.join(reversed(strs)) \* repeats)  
 return ''.join(stack)  
if \_\_name\_\_ == "\_\_main\_\_":  
 expression = "4[ac]dy"  
 solution = Solution()  
 print("输入的表达式是：", expression)  
 print("展开的字符串是：", solution.expressionExpand(expression))

### 【例96】最小子串覆盖

#### 3.代码实现

class Solution:  
 def minWindow(self, source, target):  
 if source is None:  
 return ""  
 targetHash = self.getTargetHash(target)  
 targetUniqueChars = len(targetHash)  
 matchedUniqueChars = 0  
 hash = {}  
 n = len(source)  
 j = 0  
 minLength = n + 1  
 minWindowString = ""  
 for i in range(n):  
 while j < n and matchedUniqueChars < targetUniqueChars:  
 if source[j] in targetHash:  
 hash[source[j]] = hash.get(source[j], 0) + 1  
 if hash[source[j]] == targetHash[source[j]]:  
 matchedUniqueChars += 1  
 j += 1  
 if j - i < minLength and matchedUniqueChars == targetUniqueChars:  
 minLength = j - i  
 minWindowString = source[i:j]  
 if source[i] in targetHash:  
 if hash[source[i]] == targetHash[source[i]]:  
 matchedUniqueChars -= 1  
 hash[source[i]] -= 1  
 return minWindowString  
 def getTargetHash(self, target):  
 hash = {}  
 for c in target:  
 hash[c] = hash.get(c, 0) + 1  
 return hash  
if \_\_name\_\_ == "\_\_main\_\_":  
 source = "ADOBECODEBANC"  
 target = "ABC"  
 solution = Solution()  
 print("初始的字符串是：", source, "初始的目标值是：", target)  
 print("符合条件的最短子串是：", solution.minWindow(source, target))

### 【例97】连接两个字符串中的不同字符

#### 3.代码实现

class Solution:  
 def concatenetedString(self, s1, s2):  
 result = [a for a in s1 if a not in s2] + [b for b in s2 if b not in s1]  
 return ''.join(result)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 s1 = "aacdb"  
 s2 = "gafd"  
 solution = Solution()  
 print("输入的两个字符串是：s1=", s1, "s2=", s2)  
 print("计算的结果是：", solution.concatenetedString(s1, s2))

### 【例98】字符串加法

#### 3.代码实现

class Solution(object):

def addStrings(self, num1, num2):

res = ""

m = len(num1)

n = len(num2)

i = m - 1

j = n - 1

flag = 0

while i >= 0 or j >= 0:

a = int(num1[i]) if i >= 0 else 0

i = i - 1

b = int(num2[j]) if j >= 0 else 0

j = j - 1

sum = a + b + flag

res = str(sum % 10) + res;

flag = sum // 10

return res if flag == 0 else (str(flag) + res)

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

num1 = '123'

num2 = '45'

print("初始整数：", num1, num2)

solution = Solution()

print("结果：", solution.addStrings(num1, num2))

### 【例99】字符串乘法

#### 3.代码实现

class Solution:

def multiply(self, num1, num2):

l1, l2 = len(num1), len(num2)

l3 = l1 + l2

res = [0 for i in range(l3)]

for i in range(l1 - 1, -1, -1):

carry = 0

for j in range(l2 - 1, -1, -1):

res[i + j + 1] += carry + int(num1[i]) \* int(num2[j])

carry = res[i + j + 1] // 10

res[i + j + 1] %= 10

res[i] = carry

i = 0

while i < l3 and res[i] == 0:

i += 1

res = res[i:]

return '0' if not res else ''.join(str(i) for i in res)

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

num1 = '123'

num2 = '45'

print("初始整数：", num1, num2)

solution = Solution()

print("结果：", solution.multiply(num1, num2))

### 【例100】前k个偶数长度的回文数之和

#### 3.代码实现

class Solution:

def sumKEven(self, k):

total = 0

for number in range(1, k + 1):

total += self.makePalindromeNumber(number)

return total

def makePalindromeNumber(self, number):

number\_s = str(number)

number\_s = number\_s + number\_s[::-1]

return int(number\_s)

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

k = 10

print("初始值：", k)

solution = Solution()

print("结果：", solution.sumKEven(k))

### 【例101】分割回文串Ⅰ

#### 3.代码实现

class Solution:

def minCut(self, s):

n = len(s)

f = []

p = [[False for x in range(n)] for x in range(n)]

# the worst case is cutting by each char

for i in range(n + 1):

f.append(n - 1 - i) # the last one, f[n]=-1

for i in reversed(range(n)):

for j in range(i, n):

if (s[i] == s[j] and (j - i < 2 or p[i + 1][j - 1])):

p[i][j] = True

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

return f[0]

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

s = "aab"

print("初始字符串：", s)

solution = Solution()

print("分割次数：", solution.minCut(s))

### 【例102】分割回文串Ⅱ

#### 3.代码实现

class Solution:

def partition(self, s):

results = []

self.dfs(s, [], results)

return results

def dfs(self, s, stringlist, results):

if len(s) == 0:

results.append(stringlist)

# results.append(list(stringlist))

return

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

prefix = s[:i]

if self.is\_palindrome(prefix):

self.dfs(s[i:], stringlist + [prefix], results)

# stringlist.append(prefix)

# self.dfs(s[i:], stringlist, results)

# stringlist.pop()

def is\_palindrome(self, s):

return s == s[::-1]

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

s = "aab"

print("字符串是：", s)

solution = Solution()

print("结果是：", solution.partition(s))

### 【例103】回文排列Ⅰ

#### 3.代码实现

class Solution:  
 def canPermutePalindrome(self, s):  
 lookup = {}  
 count = 0  
 for elem in s:  
 lookup[elem] = lookup.get(elem, 0) + 1  
 for val in lookup.values():  
 count += val % 2  
 return count <= 1  
if \_\_name\_\_ == "\_\_main\_\_":  
 s = "code"  
 solution = Solution()  
 print("输入的字符串是：s=", s)  
 print("输出的结果是：", solution.canPermutePalindrome(s))

### 【例104】回文排列Ⅱ

#### 3.代码实现

class Solution:

def generatePalindromes(self, s):

if not s:

return []

map = {}

for c in s:

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

if len(map.keys()) == 1:

return [s]

oddCount = 0

odd = ''

even = ''

for key, val in map.items():

if val % 2 == 1:

odd = key

oddCount += 1

even += key \* (val // 2)

if oddCount > 1:

return []

ans = []

self.permutation('', even, ans, odd)

return ans

def permutation(self, substring, s, ans, odd):

if not s:

ans.append(substring + odd + substring[::-1])

return

for i in range(len(s)):

self.permutation(substring + s[i], s[:i] + s[i + 1:], ans, odd)

#主函数

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

s = "aabb"

S = "abc"

solution = Solution()

print("s = ", s, ",结果是：", solution.generatePalindromes(s))

print("s = ", S, ",结果是：", solution.generatePalindromes(S))

#### 4.运行结果

s = aabb ,结果是：['abba', 'baab']

s = abc ,结果是：[]

### 【例105】回文链表

#### 3.代码实现

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("初始的链表是：", [node1.val, node2.val, node3.val])  
 print("最终的结果是：", solution.isPalindrome(node1))

### 【例106】有效回文串

#### 3.代码实现

class Solution:

def isPalindrome(self, s):

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

while start < end:

while start < end and not s[start].isalpha() and not s[start].isdigit():

start += 1

while start < end and not s[end].isalpha() and not s[end].isdigit():

end -= 1

if start < end and s[start].lower() != s[end].lower():

return False

start += 1

end -= 1

return True

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

temp = Solution()

string1 = "a blame malba"

string2 = "a pencil"

print(("输入："+string1))

print(("输出："+str(temp.isPalindrome(string1))))

print(("输入："+string2))

print(("输出："+str(temp.isPalindrome(string2))))

### 【例107】回文对

#### 3.代码实现

#参数words是一个独特的单词列表  
#返回所有不同索引的组合  
class Solution:

def palindromePairs(self, words):  
 if not words:  
 return []  
 table = dict()  
 for idx, word in enumerate(words):  
 table[word] = idx  
 ans = []  
 for idx, word in enumerate(words):  
 size = len(word)  
 for i in range(size + 1):  
 leftSub = word[:i]  
 rightSub = word[i:]  
 if self.isPalindrome(leftSub):  
 reversedRight = rightSub[::-1]  
 if reversedRight in table and table[reversedRight] != idx:  
 ans.append([table[reversedRight], idx])  
 if len(rightSub) > 0 and self.isPalindrome(rightSub):  
 reversedLeft = leftSub[::-1]  
 if reversedLeft in table and table[reversedLeft] != idx:  
 ans.append([idx, table[reversedLeft]])  
 return ans  
 def isPalindrome(self, word):  
 if not word:  
 return True  
 left = 0  
 right = len(word) - 1  
 while left <= right:  
 if word[left] != word[right]:  
 return False  
 left += 1  
 right -= 1  
 return True  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 words = ["bat", "tab", "cat"]  
 # 创建对象  
 solution = Solution()  
 print("输入的数组是 ", words)  
 print("输出的结果是：", solution.palindromePairs(words))

### 【例108】前k个偶数长度的回文数之和

#### 3.代码实现

class Solution:

def sumKEven(self, k):

total = 0

for number in range(1, k+1):

total += self.makePalindromeNumber(number)

return total

def makePalindromeNumber(self, number):

number\_s = str(number)

number\_s = number\_s + number\_s[::-1]

return int(number\_s)

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

temp = Solution()

nums1 = 6

nums2 = 14

print ("输入："+str(nums1))

print ("输出："+str(temp.sumKEven(nums1)))

print ("输入："+str(nums2))

print ("输出："+str(temp.sumKEven(nums2)))

### 【例109】k组翻转链表

#### 3.代码实现

class ListNode(object):  
 def \_\_init\_\_(self, val, next=None):  
 self.val = val  
 self.next = next  
class Solution:  
 def reverse(self, start, end):  
 newhead = ListNode(0)  
 newhead.next = start  
 while newhead.next != end:  
 tmp = start.next  
 start.next = tmp.next  
 tmp.next = newhead.next  
 newhead.next = tmp  
 return [end, start]  
 def reverseKGroup(self, head, k):  
 if head == None: return None  
 nhead = ListNode(0)  
 nhead.next = head  
 start = nhead  
 while start.next:  
 end = start  
 for i in range(k - 1):  
 end = end.next  
 if end.next == None: return nhead.next  
 res = self.reverse(start.next, end.next)  
 start.next = res[0]  
 start = res[1]  
 return nhead.next  
#主函数  
if \_\_name\_\_ == '\_\_main\_\_':  
 node1 = ListNode(1)  
 node2 = ListNode(2)  
 node3 = ListNode(3)  
 node4 = ListNode(4)  
 node5 = ListNode(5)  
 node1.next = node2  
 node2.next = node3  
 node3.next = node4  
 node4.next = node5  
 k = 2  
 list1 = []  
 #创建对象  
 solution = Solution()  
 newlist = solution.reverseKGroup(node1, 2)  
 while (newlist):  
 list1.append(newlist.val)  
 newlist = newlist.next  
 print("初始化的链表是：", [node1.val, node2.val, node3.val, node4.val, node5.val])  
 print(" 翻转后的结果是:", list1)

### 【例110】删除排序链表中的重复元素Ⅰ

#### 3.代码实现

class ListNode(object):  
 def \_\_init\_\_(self, val, next=None):  
 self.val = val  
 self.next = next  
class Solution:  
 def deleteDuplicates(self, head):  
 if None == head or None == head.next:  
 return head  
 new\_head = ListNode(-1)  
 new\_head.next = head  
 parent = new\_head  
 cur = head  
 while None != cur and None != cur.next: ### check cur.next None  
 if cur.val == cur.next.val:  
 val = cur.val  
 while None != cur and val == cur.val: ### check cur None  
 cur = cur.next  
 parent.next = cur  
 else:  
 cur = cur.next  
 parent = parent.next  
 return new\_head.next  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 node1 = ListNode(1)  
 node2 = ListNode(1)  
 node3 = ListNode(1)  
 node4 = ListNode(2)  
 node5 = ListNode(3)  
 node1.next = node2  
 node2.next = node3  
 node3.next = node4  
 node4.next = node5  
 list1 = []  
  
 solution = Solution()  
 print("初始链表：", [node1.val, node2.val, node3.val, node4.val, node5.val])  
 newlist = solution.deleteDuplicates(node1)  
 while (newlist):  
 list1.append(newlist.val)  
 newlist = newlist.next  
 print("删除重复元素后的结果是：", list1)

删除重复元素后的结果是：[2, 3]

### 【例111】删除排序链表中的重复元素Ⅱ

#### 3.代码实现

class ListNode(object):  
 def \_\_init\_\_(self, val, next=None):  
 self.val = val  
 self.next = next  
class Solution:  
 def deleteDuplicates(self, head):  
 delflag = 1  
 flag = 1  
 p = head  
 while (p != None and p.next != None):  
 if p.val != p.next.val:  
 flag = 1  
 p = p.next  
 elif flag < delflag:  
 flag += 1;  
 p = p.next  
 else:  
 p.next = p.next.next  
 return head  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 node1 = ListNode(1)  
 node2 = ListNode(1)  
 node3 = ListNode(2)  
 node1.next = node2  
 node2.next = node3  
 list1 = []  
  
 solution = Solution()  
 print("初始链表是：", [node1.val, node2.val, node3.val])  
 newlist = solution.deleteDuplicates(node1)  
 while (newlist):  
 list1.append(newlist.val)  
 newlist = newlist.next  
 print("删除重复元素后的链表：", list1)

### 【例112】链表划分

#### 3.代码实现

class ListNode(object):  
 def \_\_init\_\_(self, val, next=None):  
 self.val = val  
 self.next = next  
class Solution:  
 def partition(self, head, x):  
 if head is None:  
 return head  
 aHead, bHead = ListNode(0), ListNode(0)  
 aTail, bTail = aHead, bHead  
 while head is not None:  
 if head.val < x:  
 aTail.next = head  
 aTail = aTail.next  
 else:  
 bTail.next = head  
 bTail = bTail.next  
 head = head.next  
 bTail.next = None  
 aTail.next = bHead.next  
 return aHead.next  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 node1 = ListNode(1)  
 node2 = ListNode(4)  
 node3 = ListNode(3)  
 node4 = ListNode(2)  
 node5 = ListNode(5)  
 node6 = ListNode(2)  
 node1.next = node2  
 node2.next = node3  
 node3.next = node4  
 node4.next = node5  
 node5.next = node6  
 list1 = []  
 x = 3  
  
 solution = Solution()  
 print("初始链表：", [node1.val, node2.val, node3.val, node4.val, node5.val, node6.val])  
 newlist = solution.partition(node1, 3)  
 while (newlist):  
 list1.append(newlist.val)  
 newlist = newlist.next  
 print("最终的链表是：", list1)

### 【例113】翻转链表Ⅰ

#### 3.代码实现

class ListNode(object):  
 def \_\_init\_\_(self, val, next=None):  
 self.val = val  
 self.next = next  
class Solution:  
 def reverse(self, head):  
 # curt表示前继节点  
 curt = None  
 while head != None:  
 # temp记录下一个节点，head是当前节点  
 temp = head.next  
 head.next = curt  
 curt = head  
 head = temp  
 return curt  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 node1 = ListNode(1)  
 node2 = ListNode(2)  
 node3 = ListNode(3)  
 node4 = ListNode(4)  
 node1.next = node2  
 node2.next = node3  
 node3.next = node4  
 list1 = []  
   
 solution = Solution()  
 print("输入的初始链表是：", [node1.val, node2.val, node3.val, node4.val])  
 newlist = solution.reverse(node1)  
 while (newlist):  
 list1.append(newlist.val)  
 newlist = newlist.next  
 print("翻转链表后的结果是：", list1)

### 【例114】翻转链表Ⅱ

#### 3.代码实现

class ListNode(object):  
 def \_\_init\_\_(self, val, next=None):  
 self.val = val  
 self.next = next  
class Solution:  
 def reverseBetween(self, head, m, n):  
 if head is None:  
 return  
 sub\_vals = [] # contain the vals from m to n  
 dummy = ListNode(0, head)  
 fake\_head = dummy  
 i = 0  
 while fake\_head:  
 # find the m - 1 node  
 if i == m - 1:  
 cur = fake\_head.next  
 j = i + 1  
 # extract the values of the nodes ranged from m to n  
 while j >= m and j <= n:  
 # print(cur.val)  
 sub\_vals.append(cur.val)  
 cur = cur.next  
 j += 1  
 # build up reversed linked list  
 sub\_vals.reverse()  
 sub\_head = ListNode(sub\_vals[0])  
 sub\_dummy = ListNode(0, sub\_head)  
 for val in sub\_vals[1:]:  
 node = ListNode(val)  
 sub\_head.next = node  
 sub\_head = sub\_head.next  
 # relink the original list to the sub list  
 fake\_head.next = sub\_dummy.next  
 sub\_head.next = cur  
 fake\_head = fake\_head.next  
 i += 1  
 return dummy.next  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 node1 = ListNode(1)  
 node2 = ListNode(2)  
 node3 = ListNode(3)  
 node4 = ListNode(4)  
 node5 = ListNode(5)  
 node1.next = node2  
 node2.next = node3  
 node3.next = node4  
 node4.next = node5  
 list1 = []  
 m = 2  
 n = 4  
   
 solution = Solution()  
 print("初始链表是：", [node1.val, node2.val, node3.val, node4.val, node5.val], "初始的m=", m , "n=", n)  
 newlist = solution.reverseBetween(node1, m, n)  
 while (newlist):  
 list1.append(newlist.val)  
 newlist = newlist.next  
 print("翻转后的链表是：", list1)

### 【例115】旋转链表

#### 3.代码实现

class ListNode(object):  
 def \_\_init\_\_(self, val, next=None):  
 self.val = val  
 self.next = next  
class Solution:  
 def rotateRight(self, head, k):  
 if head == None:  
 return head  
 curNode = head  
 size = 1  
 while curNode != None:  
 size += 1  
 curNode = curNode.next  
 size -= 1  
 k = k % size  
 if k == 0:  
 return head  
 len = 1  
 curNode = head  
 while len < size - k:  
 len += 1  
 curNode = curNode.next  
 newHead = curNode.next  
 curNode.next = None  
 curNode = newHead  
 while curNode.next != None:  
 curNode = curNode.next  
 curNode.next = head  
 return newHead  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 node1 = ListNode(1)  
 node2 = ListNode(2)  
 node3 = ListNode(3)  
 node4 = ListNode(4)  
 node5 = ListNode(5)  
 node1.next = node2  
 node2.next = node3  
 node3.next = node4  
 node4.next = node5  
 k = 2  
 list1 = []  
  
 solution = Solution()  
 print("初始链表：", [node1.val, node2.val, node3.val, node4.val, node5.val],"初始的k=",k)  
 newlist = solution.rotateRight(node1, k)  
 while (newlist):  
 list1.append(newlist.val)  
 newlist = newlist.next  
 print("旋转后的链表是：", list1)

### 【例116】两两交换链表中的节点

#### 3.代码实现

class ListNode(object):  
 def \_\_init\_\_(self, val, next=None):  
 self.val = val  
 self.next = next  
class Solution:  
 def swapPairs(self, head):  
 if not head or not head.next: return head  
 temp = head.next  
 head.next = self.swapPairs(head.next.next)  
 temp.next = head  
 return temp  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 node1 = ListNode(1)  
 node2 = ListNode(2)  
 node3 = ListNode(3)  
 node4 = ListNode(4)  
 node1.next = node2  
 node2.next = node3  
 node3.next = node4  
 list1 = []  
 #创建对象  
 solution = Solution()  
 print("初始链表是：", [node1.val, node2.val, node3.val, node4.val])  
 newlist = solution.swapPairs(node1)  
 while (newlist):  
 list1.append(newlist.val)  
 newlist = newlist.next  
 print("交换后的链表是：", list1)

### 【例117】删除链表中的元素

#### 3.代码实现

class ListNode(object):  
 def \_\_init\_\_(self, val, next=None):  
 self.val = val  
 self.next = next  
class Solution:  
 def removeElements(self, head, val):  
 if head == None:  
 return head  
 dummy = ListNode(0)  
 dummy.next = head  
 pre = dummy  
 while head:  
 if head.val == val:  
 pre.next = head.next  
 head = pre  
 pre = head  
 head = head.next  
 return dummy.next  
 # 主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 node1 = ListNode(1)  
 node2 = ListNode(2)  
 node3 = ListNode(3)  
 node4 = ListNode(3)  
 node5 = ListNode(4)  
 node6 = ListNode(5)  
 node7 = ListNode(3)  
 node1.next=node2  
 node2.next=node3  
 node3.next=node4  
 node4.next=node5  
 node5.next=node6  
 node6.next=node7  
 val = 3  
 list1 = []  
 # 创建对象  
 solution = Solution()  
 print("初始链表是：", [node1.val, node2.val, node3.val, node4.val, node5.val, node6.val, node7.val], "需要被删除的节点val=", val)  
 newlist = solution.removeElements(node1, val)  
 while (newlist):  
 list1.append(newlist.val)  
 newlist = newlist.next  
 print("删除指定节点后的链表是：", list1)

### 【例118】重排链表

#### 3.代码实现

class ListNode(object):  
 def \_\_init\_\_(self, val, next=None):  
 self.val = val  
 self.next = next  
class Solution:  
 def reorderList(self, head):  
 if None == head or None == head.next:  
 return head  
 pfast = head  
 pslow = head  
 while pfast.next and pfast.next.next:  
 pfast = pfast.next.next  
 pslow = pslow.next  
 pfast = pslow.next  
 pslow.next = None  
 pnext = pfast.next  
 pfast.next = None  
 while pnext:  
 q = pnext.next  
 pnext.next = pfast  
 pfast = pnext  
 pnext = q  
 tail = head  
 while pfast:  
 pnext = pfast.next  
 pfast.next = tail.next  
 tail.next = pfast  
 tail = tail.next.next  
 pfast = pnext  
 return head  
#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  
 list1 = []  
 #创建对象  
 solution = Solution()  
 print("初始链表：", [node1.val, node2.val, node3.val, node4.val])  
 newlist = solution.reorderList(node1)  
 while (newlist):  
 list1.append(newlist.val)  
 newlist = newlist.next  
 print("重排后的链表是：", list1)

### 【例119】链表插入排序

#### 3.代码实现

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("插入排序后的链表是：", list1)

### 【例120】合并k个排序链表

#### 3.代码实现

# ListNode的定义

class ListNode(object):

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

self.val = val

self.next = next

class Solution:

def mergeKLists(self, lists):

self.heap = [[i, lists[i].val] for i in range(len(lists)) if lists[i] != None]

self.hsize = len(self.heap)

for i in range(self.hsize - 1, -1, -1):

self.adjustdown(i)

nHead = ListNode(0)

head = nHead

while self.hsize > 0:

ind, val = self.heap[0][0], self.heap[0][1]

head.next = lists[ind]

head = head.next

lists[ind] = lists[ind].next

if lists[ind] is None:

self.heap[0] = self.heap[self.hsize - 1]

self.hsize = self.hsize - 1

else:

self.heap[0] = [ind, lists[ind].val]

self.adjustdown(0)

return nHead.next

def adjustdown(self, p):

lc = lambda x: (x + 1) \* 2 - 1

rc = lambda x: (x + 1) \* 2

while True:

np, pv = p, self.heap[p][1]

if lc(p) < self.hsize and self.heap[lc(p)][1] < pv:

np, pv = lc(p), self.heap[lc(p)][1]

if rc(p) < self.hsize and self.heap[rc(p)][1] < pv:

np = rc(p)

if np == p:

break

else:

self.heap[np], self.heap[p] = self.heap[p], self.heap[np]

p = np

#打印链表函数

def printlist(node):

out = []

if node is None:

print(out)

while node.next is not None:

out.append(node.val)

node = node.next

out.append(node.val)

print(out)

#主函数

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

node1 = ListNode(2)

node2 = ListNode(4)

node3 = ListNode(-1)

node1.next = node2

#创建对象

solution = Solution()

A = [node1, node3]

print("排序后的链表是：", end="")

printlist(solution.mergeKLists(A))

### 【例121】带环链表Ⅰ

#### 3.代码实现

class ListNode(object):  
 def \_\_init\_\_(self, val, next=None):  
 self.val = val  
 self.next = next  
class Solution:  
 def hasCycle(self, head):  
 if not head or not head.next:  
 return False  
 slow = head  
 fast = head  
 while fast and fast.next:  
 slow = slow.next  
 fast = fast.next.next  
 if slow is fast:  
 return True  
 return False  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 node1 = ListNode(-21)  
 node2 = ListNode(10)  
 node3 = ListNode(4)  
 node4 = ListNode(5)  
 node1.next = node2  
 node2.next = node3  
 node3.next = node4  
 node4.next = node1  
 # 创建对象  
 solution = Solution()  
 print("初始化的值是：", [node1.val, node2.val, node3.val, node4.val])  
 print("结果是：", solution.hasCycle(node1))

### 【例122】带环链表II

#### 3.代码实现

class ListNode(object):  
 def \_\_init\_\_(self, val, next=None):  
 self.val = val  
 self.next = next  
class Solution:  
 def detectCycle(self, head):  
 if not head or not head.next:  
 return None  
 fast, slow = head, head  
 while fast and fast.next:  
 slow = slow.next  
 fast = fast.next.next  
 if slow is fast:  
 break  
 if fast and slow is fast:  
 slow = head  
 while slow is not fast:  
 slow = slow.next  
 fast = fast.next  
 return slow  
 return None  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 node1 = ListNode(-21)  
 node2 = ListNode(10)  
 node3 = ListNode(4)  
 node4 = ListNode(5)  
 node1.next = node2  
 node2.next = node3  
 node3.next = node4  
 node4.next = node2  
 #创建对象  
 solution = Solution()  
 print("初始链表：", [node1.val, node2.val, node3.val, node4.val])  
 print("结果是：", solution.detectCycle(node1).val)

### 【例123】删除链表中倒数第n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(object):  
 def removeNthFromEnd(self, head, n):  
 res = ListNode(0)  
 res.next = head  
 tmp = res  
 for i in range(0, n):  
 head = head.next  
 while head != None:  
 head = head.next  
 tmp = tmp.next  
 tmp.next = tmp.next.next  
 return res.next  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 node1 = ListNode(1)  
 node2 = ListNode(2)  
 node3 = ListNode(3)  
 node4 = ListNode(4)  
 node5 = ListNode(5)  
 node1.next = node2  
 node2.next = node3  
 node3.next = node4  
 node4.next = node5  
 list1 = []  
 n = 2  
 #创建对象  
 solution = Solution()  
 print("初始链表是：", [node1.val, node2.val, node3.val, node4.val, node5.val])  
 newlist = solution.removeNthFromEnd(node1, n)  
 while (newlist):  
 list1.append(newlist.val)  
 newlist = newlist.next  
 print("最终链表是：", list1)

### 【例124】链表排序

#### 3.代码实现

class ListNode(object):  
 def \_\_init\_\_(self, val, next=None):  
 self.val = val  
 self.next = next  
class Solution:  
 def sortList(self, head):  
 def merge(list1, list2):  
 if list1 == None:  
 return list2  
 if list2 == None:  
 return list1  
 head = None  
 if list1.val < list2.val:  
 head = list1  
 list1 = list1.next  
 else:  
 head = list2;  
 list2 = list2.next;  
 tmp = head  
 while list1 != None and list2 != None:  
 if list1.val < list2.val:  
 tmp.next = list1  
 tmp = list1  
 list1 = list1.next  
 else:  
 tmp.next = list2  
 tmp = list2  
 list2 = list2.next  
 if list1 != None:  
 tmp.next = list1;  
 if list2 != None:  
 tmp.next = list2;  
 return head;  
 if head == None:  
 return head  
 if head.next == None:  
 return head  
 fast = head  
 slow = head  
 while fast.next != None and fast.next.next != None:  
 fast = fast.next.next  
 slow = slow.next  
 mid = slow.next  
 slow.next = None  
 list1 = self.sortList(head)  
 list2 = self.sortList(mid)  
 sorted = merge(list1, list2)  
 return sorted  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 node1 = ListNode(1)  
 node2 = ListNode(3)  
 node3 = ListNode(2)  
 node1.next = node2  
 node2.next = node3  
 list1 = []  
 #创建对象  
 solution = Solution()  
 print("初始链表是：", [node1.val, node2.val, node3.val])  
 newlist = solution.sortList(node1)  
 while (newlist):  
 list1.append(newlist.val)  
 newlist = newlist.next  
 print("排序后的链表：", list1)

### 【例125】加一链表

#### 3.代码实现

#创建链表  
#head的类型是链表节点  
#返回值的类型是链表节点  
class ListNode(object):  
 def \_\_init\_\_(self, val, next=None):  
 self.val = val  
 self.next = next  
class Solution(object):  
 def plusOne(self, head):  
 stack = []  
 h = head  
 while h:  
 stack.append(h)  
 h = h.next  
 while stack and stack[-1].val == 9:  
 stack[-1].val = 0  
 stack.pop()  
 if stack:  
 stack[-1].val += 1  
 else:  
 node = ListNode(1)  
 node.next = head  
 head = node  
 return head  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 node1 = ListNode(1)  
 node2 = ListNode(2)  
 node3 = ListNode(3)  
 node1.next = node2  
 node2.next = node3  
 list1 = []  
 #创建对象  
 solution = Solution()  
 print("初始链表是：", [node1.val, node2.val, node3.val])  
 newlist = solution.plusOne(node1)  
 while (newlist):  
 list1.append(newlist.val)  
 newlist = newlist.next  
 print("最终得到的链表是：", list1)

### 【例126】交换链表当中两个节点

#### 3.代码实现

class ListNode(object):  
 def \_\_init\_\_(self, val, next=None):  
 self.val = val  
 self.next = next  
class Solution:  
 def swapNodes(self, head, v1, v2):  
 dummy = ListNode(0, head)  
 cur = dummy  
 p1 = None  
 p2 = None  
 while cur.next != None:  
 if cur.next.val == v1:  
 p1 = cur  
 if cur.next.val == v2:  
 p2 = cur  
 cur = cur.next  
 if p1 is None or p2 is None:  
 return dummy.next  
 n1 = p1.next  
 n2 = p2.next  
 n1next = n1.next  
 n2next = n2.next  
 if p1.next == p2:  
 p1.next = n2  
 n2.next = n1  
 n1.next = n2next  
 elif p2.next == p1:  
 p2.next = n1  
 n1.next = n2  
 n2.next = n1next  
 else:  
 p1.next = n2  
 n2.next = n1next  
 p2.next = n1  
 n1.next = n2next  
 return dummy.next  
#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  
 v1 = 2  
 v2 = 4  
 list1 = []  
 #创建对象  
 solution = Solution()  
 print("初始的链表是：", [node1.val, node2.val, node3.val, node4.val], "初始的两个权值v1=", v1, "v2=", v2)  
 newlist = solution.swapNodes(node1, v1, v2)  
 while (newlist):  
 list1.append(newlist.val)  
 newlist = newlist.next  
 print("交换后的链表结果是：", list1)

### 【例127】线段树的修改

#### 3.代码实现

#树的定义  
class TreeNode:  
 def \_\_init\_\_(self, start, end, max):  
 self.max = max  
 self.start = start  
 self.end = end  
 self.left, self.right = None, None  
class Solution:  
#参数root、index、value是线段树的根，并使用[index, index]将节点的值更改为新的给定值  
#返回列表  
 def modify(self, root, index, value):  
 if root is None:  
 return  
 if root.start == root.end:  
 root.max = value  
 return  
 if root.left.end >= index:  
 self.modify(root.left, index, value)  
 else:  
 self.modify(root.right, index, value)  
 root.max = max(root.left.max, root.right.max)  
if \_\_name\_\_ == '\_\_main\_\_':  
 #构建树  
 root = TreeNode(1, 4, 3)  
 root.left = TreeNode(1, 2, 2)  
 root.right = TreeNode(3, 4, 3)  
 root.left.left = TreeNode(1, 1, 2)  
 root.left.right = TreeNode(2, 2, 1)  
 root.right.left = TreeNode(3, 3, 0)  
 root.right.right = TreeNode(4, 4, 3)  
 solution = Solution()  
 print("调用modify(root,2,4)")  
 solution.modify(root, 2, 4)  
 print([root.max, root.left.max, root.right.max, root.left.left.max, root.left.right.max, root.right.left.max,  
 root.right.right.max])  
 print("调用modify(root,4,0)")  
 solution.modify(root, 4, 0)  
 print([root.max, root.left.max, root.right.max, root.left.left.max, root.left.right.max, root.right.left.max, root.right.right.max])

### 【例128】线段树的构造Ⅰ

#### 3.代码实现

#参数A是一个整数数组  
#返回线段树的根  
class SegmentTreeNode:  
 def \_\_init\_\_(self, start, end, max):  
 self.start, self.end, self.max = start, end, max  
 self.left, self.right = None, None  
class Solution:  
 def build(self, A):  
 return self.buildTree(0, len(A) - 1, A)  
 def buildTree(self, start, end, A):  
 if start > end:  
 return None  
 node = SegmentTreeNode(start, end, A[start])  
 if start == end:  
 return node  
 mid = (start + end) // 2  
 node.left = self.buildTree(start, mid, A)  
 node.right = self.buildTree(mid + 1, end, A)  
 if node.left is not None and node.left.max > node.max:  
 node.max = node.left.max  
 if node.right is not None and node.right.max > node.max:  
 node.max = node.right.max  
 return node  
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.max)  
 res.append(tmp)  
 print(res)  
if \_\_name\_\_ == '\_\_main\_\_':  
 A = [3, 2, 1, 4]  
 print("输入的数组是：", A)  
 solution = Solution()  
 root = solution.build(A)  
 print("构造的线段树是：")  
 printTree(root)

### 【例129】线段树的构造II

#### 3.代码实现

class SegmentTreeNode:  
 def \_\_init\_\_(self, start, end):  
 self.start, self.end = start, end  
 self.left, self.right = None, None  
class Solution:  
#参数start、end表示段/区间  
#返回线段树的根节点  
 def build(self, start, end):  
 if start > end:  
 return None  
 root = SegmentTreeNode(start, end)  
 if start == end:  
 return root  
 root.left = self.build(start, (start + end) // 2)  
 root.right = self.build((start + end) // 2 + 1, end)  
 return root  
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.start)  
 tmp.append(r.end)  
 res.append(tmp)  
 print(res)  
if \_\_name\_\_ == '\_\_main\_\_':  
 solution = Solution()  
 root = solution.build(1, 6)  
 print("构造的线段树是：")  
 printTree(root)

### 【例130】线段树查询Ⅰ

#### 3.代码实现

#线段树的定义  
class SegmentTreeNode:  
 def \_\_init\_\_(self, start, end, count):  
 self.start, self.end, self.count = start, end, count  
 self.left, self.right = None, None  
class Solution:  
#参数root、start、end是线段树的根节点，一个段和间隔  
#返回值是间隔[start, end]中计数的元素个数  
 def query(self, root, start, end):  
 if root is None:  
 return 0  
 if root.start > end or root.end < start:  
 return 0  
 if start <= root.start and root.end <= end:  
 return root.count  
 return self.query(root.left, start, end) + self.query(root.right, start, end)  
#主函数  
if \_\_name\_\_ == '\_\_main\_\_':  
 root = SegmentTreeNode(0, 3, 3)  
 root.left = SegmentTreeNode(0, 1, 1)  
 root.right = SegmentTreeNode(2, 3, 2)  
 root.left.left = SegmentTreeNode(0, 0, 1)  
 root.left.right = SegmentTreeNode(1, 1, 0)  
 root.right.left = SegmentTreeNode(2, 2, 1)  
 root.right.right = SegmentTreeNode(3, 3, 1)  
 solution = Solution()  
 print("对于数组[0,null,2,3]的线段树，查询为（1,1）的结果是：", solution.query(root, 1, 1))

### 【例131】线段树查询Ⅱ

#### 3.代码实现

class SegmentTreeNode:  
 def \_\_init\_\_(self, start, end, max):  
 self.start, self.end, self.max = start, end, max  
 self.left, self.right = None, None  
class Solution:  
#参数A是一个整数数组  
#返回值是线段树的根节点  
 def build(self, A):  
 return self.buildTree(0, len(A) - 1, A)  
 def buildTree(self, start, end, A):  
 if start > end:  
 return None  
 node = SegmentTreeNode(start, end, A[start])  
 if start == end:  
 return node  
 mid = (start + end) // 2  
 node.left = self.buildTree(start, mid, A)  
 node.right = self.buildTree(mid + 1, end, A)  
 if node.left is not None and node.left.max > node.max:  
 node.max = node.left.max  
 if node.right is not None and node.right.max > node.max:  
 node.max = node.right.max  
 return node  
 def query(self, root, start, end):  
 #   
 if root.start > end or root.end < start:  
 return -0x7fffff  
 if start <= root.start and root.end <= end:  
 return root.max  
 return max(self.query(root.left, start, end), \  
 self.query(root.right, start, end))  
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.max)  
 res.append(tmp)  
 print(res)  
if \_\_name\_\_ == '\_\_main\_\_':  
 A = [1, 4, 2, 3]  
 print("输入的数组是：", A)  
 solution = Solution()  
 root = solution.build(A)  
 print("构造的线段树是：")  
 printTree(root)  
 print("运行query(root,1,1):", solution.query(root, 1, 1))  
 print("运行query(root,1,2):", solution.query(root, 1, 2))  
 print("运行query(root,2,3):", solution.query(root, 2, 3))  
 print("运行query(root,0,2):", solution.query(root, 0, 2))

### 【例132】是否为子树

#### 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T1、T2是二叉树的根节点

#返回值是一个布尔值，当T2是T1的子树时返回True，否则返回False

def get(self, root, rt):

if root is None:

rt.append("#")

return

rt.append(str(root.val))

self.get(root.left, rt)

self.get(root.right, rt)

def isSubtree(self, T1, T2):

rt = []

self.get(T1, rt)

t1 = ','.join(rt)

rt = []

self.get(T2, rt)

t2 = ','.join(rt)

return t1.find(t2) != -1

#主函数

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

root1 = TreeNode(1)

root1.left = TreeNode(2)

root1.right = TreeNode(3)

root1.right.left = TreeNode(4)

root2 = TreeNode(3)

root2.left = TreeNode(4)

solution = Solution()

print("T2是否是T1的子树：", solution.isSubtree(root1, root2))

### 【例133】最小子树

#### 3.代码实现

#树的定义

class TreeNode:

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

self.val = val

self.left, self.right = None, None

import sys

class Solution:

def findSubtree(self, root):

self.minumum\_weight = sys.maxsize

self.result = None

self.helper(root)

return self.result

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.minumum\_weight:

self.minumum\_weight = left\_weight + right\_weight + root.val

self.result = root

return left\_weight + right\_weight + root.val

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(2)

root.left.left = TreeNode(0)

root.left.right = TreeNode(2)

root.right.left = TreeNode(-4)

root.right.right = TreeNode(-5)

solution = Solution()

print("最小子树的根节点是：", solution.findSubtree(root).val)

### 【例134】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:

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("给定二叉树是：", printTree(root))

print("最大平均值的子树是：", printTree(solution.findSubtree2(root)))

### 【例135】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:  
 def closestKValues(self, root, target, k):  
 if root is None:  
 return []  
 self.allnodes = {}  
 diff = []  
 result = []  
 self.dsf(root, target)  
 for i in self.allnodes:  
 diff.append(i)  
 diff.sort()  
 for i in range(k):  
 a = diff[i]  
 result.append(self.allnodes[a])  
 return result  
 def dsf(self, root, target):  
 if root is None:  
 return  
 dif = abs(target - root.val)  
 self.allnodes[dif] = root.val  
 self.dsf(root.left, target)  
 self.dsf(root.right, target)  
 return  
#主函数  
if \_\_name\_\_=="\_\_main\_\_":  
 root=TreeNode(1)  
 target = 0.000000  
 k = 1  
 #创建对象  
 solution=Solution()  
 print("root=",root,"target=",target,"k=",k)  
 print("输出的结果是：",solution.closestKValues(root,target,k))

### 【例136】二叉搜索树中插入节点

#### 3.代码实现

class TreeNode:  
 def \_\_init\_\_(self, val):  
 self.val = val  
 self.left, self.right = None, None  
class Solution:  
 def insertNode(self, root, node):  
 if root is None:  
 return node  
 curt = root  
 while curt != node:  
 if node.val < curt.val:  
 if curt.left is None:  
 curt.left = node  
 curt = curt.left  
 else:  
 if curt.right is None:  
 curt.right = node  
 curt = curt.right  
 return root  
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)  
 print(res)  
if \_\_name\_\_ == '\_\_main\_\_':  
 root = TreeNode(2)  
 root.left = TreeNode(1)  
 root.right = TreeNode(4)  
 root.right.left = TreeNode(3)  
 solution = Solution()  
 node = TreeNode(6)  
 print("原始二叉树为")  
 printTree(root)  
 print("插入节点为：")  
 printTree(node)  
 root0 = solution.insertNode(root, node)  
 print("插入后的树为")  
 printTree(root0)

### 【例137】二叉搜索树中删除节点

#### 3.代码实现

class TreeNode:  
 def \_\_init\_\_(self, val):  
 self.val = val  
 self.left, self.right = None, None  
class Solution:  
 ans = []  
 def inorder(self, root, value):  
 if root is None:  
 return  
 self.inorder(root.left, value)  
 if root.val != value:  
 self.ans.append(root.val)  
 self.inorder(root.right, value)  
 def build(self, l, r):  
 if l == r:  
 node = TreeNode(self.ans[l])  
 return node  
 if l > r:  
 return None  
 mid = (l + r) // 2 + 1  
 node = TreeNode(self.ans[mid])  
 node.left = self.build(l, mid - 1)  
 node.right = self.build(mid + 1, r)  
 return node  
 def removeNode(self, root, value):  
 self.inorder(root, value)  
 return self.build(0, len(self.ans) - 1)  
#遍历这个树  
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)  
#遍历这个二叉树  
def inorderTraversal(root):  
 if root == None:  
 return []  
 res = []  
 res += inorderTraversal(root.left)  
 res.append(root.val)  
 res += inorderTraversal(root.right)  
 return res  
# 主函数  
if \_\_name\_\_ == '\_\_main\_\_':  
 root = TreeNode(5)  
 root.left = TreeNode(3)  
 root.right = TreeNode(6)  
 root.left.left = TreeNode(2)  
 root.left.right = TreeNode(4)  
 solution = Solution()  
 print("原来的二叉树是：", inorderTraversal(root))  
 n = int(input("请输入要删除的节点值："))  
 print("删除后的二叉树是：", inorderTraversal(solution.removeNode(root, n)))

### 【例138】二叉搜索树转化成更大的树

#### 3.代码实现

# 树的定义  
#参数root是二叉树的根节点，类型是树节点  
#返回值的类型是树节点，表示新的根  
class TreeNode:  
 def \_\_init\_\_(self, val):  
 self.val = val  
 self.left, self.right = None, None  
class Solution:  
 def convertBST(self, root):   
 self.sum = 0  
 self.helper(root)  
 return root  
 def helper(self, root):  
 if root is None:  
 return  
 if root.right:  
 self.helper(root.right)  
 self.sum += root.val  
 root.val = self.sum  
 if root.left:  
 self.helper(root.left)  
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)  
 print(res)  
if \_\_name\_\_ == '\_\_main\_\_':  
 root = TreeNode(5)  
 root.left = TreeNode(2)  
 root.right = TreeNode(13)  
 solution = Solution()  
 print("原始二叉树为")  
 printTree(root)  
 root0 = solution.convertBST(root)  
 print("转换后的树为")  
 printTree(root0)

### 【例139】二叉搜索树中搜索区间

#### 3.代码实现

class TreeNode:  
 def \_\_init\_\_(self, val):  
 self.val = val  
 self.left, self.right = None, None  
class Solution:  
 def searchRange(self, root, k1, k2):  
 ans = []  
 if root is None:  
 return ans  
 queue = [root]  
 index = 0  
 while index < len(queue):  
 if queue[index] is not None:  
 if queue[index].val >= k1 and \  
 queue[index].val <= k2:  
 ans.append(queue[index].val)  
 queue.append(queue[index].left)  
 queue.append(queue[index].right)  
 index += 1  
 return sorted(ans)  
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)  
 print(res)  
if \_\_name\_\_ == '\_\_main\_\_':  
 root = TreeNode(20)  
 root.left = TreeNode(8)  
 root.right = TreeNode(22)  
 root.left.left = TreeNode(4)  
 root.left.right = TreeNode(12)  
 print("原始二叉树是：")  
 printTree(root)  
 k1 = 10  
 k2 = 22  
 print("k1=", k1, "\nk2=", k2)  
 solution = Solution()  
 print("所有升序的节点值是：", solution.searchRange(root, k1, k2))

### 【例140】二叉搜索树的中序后继

#### 3.代码实现

class TreeNode:  
 def \_\_init\_\_(self, val):  
 self.val = val  
 self.left, self.right = None, None  
class Solution:  
 def inorderSuccessor(self, root, p):  
 successor = None  
 while root:  
 if root.val > p.val:  
 successor = root  
 root = root.left  
 else:  
 root = root.right  
 return successor  
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)  
 print(res)  
if \_\_name\_\_ == '\_\_main\_\_':  
 root = TreeNode(2)  
 root.left = TreeNode(1)  
 root.right = TreeNode(3)  
 solution = Solution()  
 print("原始二叉树为")  
 printTree(root)  
 node = TreeNode(2)  
 print("给定的节点是")  
 printTree(node)  
 root0 = solution.inorderSuccessor(root, node)  
 print("该节点的后序遍历的后继是")  
 printTree(root0)

### 【例141】二叉搜索树两数之和

#### 3.代码实现

class TreeNode:  
 def \_\_init\_\_(self, val):  
 self.val = val  
 self.left, self.right = None, None  
class Solution:  
 def twoSum(self, root, n):  
 if not root:  
 return  
 stack, check = [], set()  
 while stack or root:  
 while root:  
 stack.append(root)  
 root = root.left  
 root = stack.pop()  
 if root.val == n:  
 return root.val  
 elif n - root.val in check:  
 return [root.val, n - root.val]  
 if root.val not in check:  
 check.add(root.val)  
 root = root.right  
 return False  
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)  
 print(res)  
if \_\_name\_\_ == '\_\_main\_\_':  
 root = TreeNode(4)  
 root.left = TreeNode(2)  
 root.right = TreeNode(5)  
 root.left.left = TreeNode(1)  
 root.left.right = TreeNode(3)  
 print("原始二叉树是：")  
 printTree(root)  
 n = 3  
 solution = Solution()  
 print("n=", n)  
 print("在树中找到的和为n的两个数字是：", solution.twoSum(root, n))

### 【例142】裁剪二叉搜索树

#### 3.代码实现

# 树的定义  
#参数root是二叉搜索树的根节点  
#参数minimum是最小的限制值  
#参数maximum是最大的限制值  
#返回值是新树的根节点  
class TreeNode:  
 def \_\_init\_\_(self, val):  
 self.val = val  
 self.left, self.right = None, None  
class Solution:  
 def trimBST(self, root, minimum, maximum):  
 if not root:  
 return None  
 if root.val < minimum:  
 return self.trimBST(root.right, minimum, maximum)  
 if root.val > maximum:  
 return self.trimBST(root.left, minimum, maximum)  
 root.left = self.trimBST(root.left, minimum, maximum)  
 root.right = self.trimBST(root.right, minimum, maximum)  
 return root  
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)  
 print(res)  
if \_\_name\_\_ == '\_\_main\_\_':  
 root = TreeNode(8)  
 root.left = TreeNode(3)  
 root.right = TreeNode(10)  
 root.left.left = TreeNode(1)  
 root.left.right = TreeNode(6)  
 root.right.right = TreeNode(14)  
 root.left.right.left = TreeNode(4)  
 root.left.right.right = TreeNode(7)  
 root.right.right.left = TreeNode(13)  
 solution = Solution()  
 print("原始二叉树为")  
 printTree(root)  
 min = 5  
 max = 13  
 print("给定的max和min分别是", max, min)  
 root0 = solution.trimBST(root, min, max)  
 print("二分搜索树的结果是")  
 printTree(root0)

#### 4.运行结果

原始二叉树为

[[8], [3, 10], [1, 6, 14], [4, 7, 13]]

给定的max和min分别是13、5

二分搜索树的结果是

[[8], [6, 10], [7, 13]]

### 【例143】统计完全二叉树节点数

#### 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(object):  
 def countNodes(self, root):  
#root的类型是树节点  
#返回值的类型是整数型  
#一直向左下走来计算深度  
 def getDepth(Node):  
 if Node == None:  
 return 0  
 depth = 1  
 while Node.left != None:  
 depth += 1  
 Node = Node.left  
 return depth  
 if root == None:  
 return 0  
 rightT = root.right  
 leftT = root.left  
 rDepth = getDepth(rightT)  
 lDepth = getDepth(leftT)  
 #如果左右子树深度相同，那么说明右子数是满二叉树，左子树是完全二叉树  
 if rDepth == lDepth:  
 return self.countNodes(rightT) + 2 \*\* lDepth  
 #否则说明左子树是满二叉树，右子树是完全二叉树  
 else:  
 return self.countNodes(leftT) + 2 \*\* rDepth  
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)  
 print(res)  
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)  
 solution = Solution()  
 print("原始二叉树为")  
 printTree(root)  
 total = solution.countNodes(root)  
 print("完全树节点数是：", total)

### 【例144】二叉搜索树迭代器

#### 3.代码实现

class TreeNode:  
 def \_\_init\_\_(self, val):  
 self.val = val  
 self.left, self.right = None, None  
class BSTIterator:  
#参数root是二叉树的根节点  
 def \_\_init\_\_(self, root):  
 self.stack = []  
 self.curt = root  
#如果有下一个节点就返回True，否则返回false  
 def hasNext(self):  
 return self.curt is not None or len(self.stack) > 0  
#返回下一个节点  
 def next(self):  
 while self.curt is not None:  
 self.stack.append(self.curt)  
 self.curt = self.curt.left  
 self.curt = self.stack.pop()  
 nxt = self.curt  
 self.curt = self.curt.right  
 return nxt  
if \_\_name\_\_ == '\_\_main\_\_':  
 root = TreeNode(10)  
 root.left = TreeNode(1)  
 root.right = TreeNode(11)  
 root.left.right = TreeNode(6)  
 root.right.right = TreeNode(12)  
 iterator = BSTIterator(root)  
 print("使用迭代器进行中序遍历的结果是：")  
 while iterator.hasNext():  
 node = iterator.next()  
 print(node.val)

### 【例145】翻转二叉树

#### 3.代码实现

#树的定义  
#参数root是一个树节点，表示二叉树的根节点  
#返回翻转后值  
class invertBinaryTree:  
 def \_\_init\_\_(self, val):  
 self.val = val  
 self.left, self.right = None, None  
class Solution:  
 def invertBinaryTree(self, root):  
 self.dfs(root)  
 def dfs(self, node):  
 left = node.left  
 right = node.right  
 node.left = right  
 node.right = left  
 if (left != None): self.dfs(left)  
 if (right != None): self.dfs(right)  
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)  
 print(res)  
if \_\_name\_\_ == '\_\_main\_\_':  
 root = invertBinaryTree(1)  
 root.left = invertBinaryTree(2)  
 root.right = invertBinaryTree(3)  
 root.right.left = invertBinaryTree(4)  
 print("原始二叉树为：")  
 printTree(root)  
 solution = Solution()  
 solution.invertBinaryTree(root)  
 print("翻转后的二叉树为：")  
 printTree(root)

### 【例146】相同二叉树

#### 3.代码实现

class TreeNode:  
 def \_\_init\_\_(self, val):  
 self.val = val  
 self.left, self.right = None, None  
class Solution:  
 def isIdentical(self, a, b):  
 if a == None and b == None:  
 return True  
 if a == None or b == None:  
 return False  
 if a.val != b.val:  
 return False  
 a1 = self.isIdentical(a.left, b.left)  
 b1 = self.isIdentical(a.right, b.right)  
 return a1 and b1  
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)  
 print(res)  
if \_\_name\_\_ == '\_\_main\_\_':  
 root1 = TreeNode(1)  
 root1.left = TreeNode(2)  
 root1.right = TreeNode(3)  
 root1.left.left = TreeNode(4)  
 print("原始二叉树1为")  
 printTree(root1)  
 root2 = TreeNode(1)  
 root2.left = TreeNode(2)  
 root2.right = TreeNode(3)  
 root2.left.right = TreeNode(4)  
 print("原始二叉树2为")  
 printTree(root2)  
 solution = Solution()  
 print("二叉树1与二叉树1进行判断：", solution.isIdentical(root1, root1))  
 print("二叉树1与二叉树2进行判断：", solution.isIdentical(root1, root2))

### 【例147】前序遍历和中序遍历树构造二叉树

#### 3.代码实现

class TreeNode:  
 def \_\_init\_\_(self, val):  
 self.val = val  
 self.left, self.right = None, None  
class Solution:  
 def buildTree(self, preorder, inorder):  
 if not inorder: return None # inorder is empty  
 root = TreeNode(preorder[0])  
 rootPos = inorder.index(preorder[0])  
 root.left = self.buildTree(preorder[1: 1 + rootPos], inorder[: rootPos])  
 root.right = self.buildTree(preorder[rootPos + 1:], inorder[rootPos + 1:])  
 return root  
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)  
 print(res)  
if \_\_name\_\_ == '\_\_main\_\_':  
 inorder = [1, 2, 3]  
 preorder = [2, 1, 3]  
 print("前序遍历为：", preorder)  
 print("中序遍历为：", inorder)  
 solution = Solution()  
 root = solution.buildTree(preorder, inorder)  
 print("构造的二叉树为：")  
 printTree(root)

### 【例148】二叉树的后序遍历

class TreeNode:  
 def \_\_init\_\_(self, val):  
 self.val = val  
 self.left, self.right = None, None  
class Solution:  
 result = []  
 def traverse(self, root):  
 if root is None:  
 return  
 self.traverse(root.left)  
 self.traverse(root.right)  
 self.results.append(root.val)  
 def postorderTraversal(self, root):  
 self.results = []  
 self.traverse(root)  
 return self.results  
 #   
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)  
 print(res)  
if \_\_name\_\_ == '\_\_main\_\_':  
 root = TreeNode(1)  
 root.right = TreeNode(2)  
 root.right.left = TreeNode(3)  
 print("原始二叉树为")  
 printTree(root)  
 solution = Solution()  
 print("后序遍历的结果为", solution.postorderTraversal(root))

### 【例149】二叉树的所有路径

class TreeNode:  
 def \_\_init\_\_(self, val):  
 self.val = val  
 self.left, self.right = None, None  
class Solution:  
 def binaryTreePaths(self, root):  
 if root is None:  
 return []  
 result = []  
 self.dfs(root, [], result)  
 return result  
 def dfs(self, node, path, result):  
 path.append(str(node.val))  
 if node.left is None and node.right is None:  
 result.append('->'.join(path))  
 path.pop()  
 return  
 if node.left:  
 self.dfs(node.left, path, result);  
 if node.right:  
 self.dfs(node.right, path, result)  
 path.pop()  
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)  
 print(res)  
if \_\_name\_\_ == '\_\_main\_\_':  
 root = TreeNode(1)  
 root.left = TreeNode(2)  
 root.right = TreeNode(3)  
 root.left.right = TreeNode(5)  
 print("原始二叉树为")  
 printTree(root)  
 solution = Solution()  
 print("后序遍历的结果为", solution.binaryTreePaths(root))

### 【例150】中序遍历和后序遍历树构造二叉树

class TreeNode:  
 def \_\_init\_\_(self, val):  
 self.val = val  
 self.left, self.right = None, None  
class Solution:  
 def buildTree(self, inorder, postorder):  
 if not inorder: return None  
 root = TreeNode(postorder[-1])  
 rootPos = inorder.index(postorder[-1])  
 root.left = self.buildTree(inorder[: rootPos], postorder[: rootPos])  
 root.right = self.buildTree(inorder[rootPos + 1:], postorder[rootPos: -1])  
 return root  
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)  
 print(res)  
if \_\_name\_\_ == '\_\_main\_\_':  
 inorder = [1, 2, 3]  
 postorder = [1, 3, 2]  
 print("中序遍历为：", inorder)  
 print("后序遍历为：", postorder)  
 solution = Solution()  
 root = solution.buildTree(inorder, postorder)  
 print("构造的二叉树为：")  
 printTree(root)

### 【例151】二叉树的序列化和反序列化

#### 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:  
 def serialize(self, root):  
 if not root:  
 return ['#']  
 ans = []  
 ans.append(str(root.val))  
 ans += self.serialize(root.left)  
 ans += self.serialize(root.right)  
 return ans  
 def deserialize(self, data):  
 ch = data.pop(0)  
 if ch == '#':  
 return None  
 else:  
 root = TreeNode(int(ch))  
 root.left = self.deserialize(data)  
 root.right = self.deserialize(data)  
 return root  
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)  
 print(res)  
if \_\_name\_\_ == '\_\_main\_\_':  
 root = TreeNode(3)  
 root.left = TreeNode(9)  
 root.right = TreeNode(20)  
 root.right.left = TreeNode(15)  
 root.right.right = TreeNode(7)  
 print("原始二叉树为：")  
 printTree(root)  
 solution = Solution()  
 print("将二叉树序列化：")  
 list0 = solution.serialize(root)  
 print(list0)  
 print("将序列化的数字再次反序列化：")  
 root0 = solution.deserialize(list0)  
 printTree(root0)

### 【例152】二叉树的层次遍历Ⅰ

#### 3.代码实现

from collections import deque

class TreeNode:

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

self.val = val

self.left, self.right = None, None

class Solution:

def levelOrder(self, root):

if root is None:

return []

queue = deque([root])

result = []

while queue:

level = []

for \_ in range(len(queue)):

node = queue.popleft()

level.append(node.val)

if node.left:

queue.append(node.left)

if node.right:

queue.append(node.right)

result.append(level)

return result

# 主函数

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

root = TreeNode(3)

root.left = TreeNode(9)

root.right = TreeNode(20)

root.right.left = TreeNode(15)

root.right.right = TreeNode(7)

solution = Solution()

print("层序遍历的结果是：", solution.levelOrder(root))

### 【例153】二叉树的层次遍历Ⅱ

class TreeNode:

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

self.val = val

self.left, self.right = None, None

class Solution:

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\_\_':

root = TreeNode(3)

root.left = TreeNode(9)

root.right = TreeNode(20)

root.right.left = TreeNode(15)

root.right.right = TreeNode(7)

solution = Solution()

print("层次遍历的结果是：", solution.levelOrderBottom(root))

### 【例154】二叉树的锯齿形层次遍历

class TreeNode:

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

self.val = val

self.left, self.right = None, None

class Solution:

def preorder(self, root, level, res):

if root:

if len(res) < level + 1: res.append([])

if level % 2 == 0:

res[level].append(root.val)

else:

res[level].insert(0, root.val)

self.preorder(root.left, level + 1, res)

self.preorder(root.right, level + 1, res)

def zigzagLevelOrder(self, root):

self.results = []

self.preorder(root, 0, self.results)

return self.results

# 主函数

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

root = TreeNode(3)

root.left = TreeNode(9)

root.right = TreeNode(20)

root.right.left = TreeNode(15)

root.right.right = TreeNode(7)

solution = Solution()

print("锯齿形层次遍历的结果是：", solution.zigzagLevelOrder(root))

### 【例155】寻找二叉树叶子节点

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

#返回值是收集并移除的所有叶子节点

def findLeaves(self, root):

ans = []

self.depth = {}

maxDepth = self.dfs(root)

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

ans.append(self.depth.get(i))

return ans

def dfs(self, node):

#寻找树深度

if node is None:

return 0

d = max(self.dfs(node.left), self.dfs(node.right)) + 1

if d not in self.depth:

self.depth[d] = []

self.depth[d].append(node.val)

return d

# 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("收集的节点是：", solution.findLeaves(root))

### 【例156】平衡二叉树

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157】二叉树中的最大路径和

class TreeNode:

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

self.val = val

self.left, self.right = None, None

class Solution:

def maxPathSum(self, root):

self.maxSum = float('-inf')

self.\_maxPathSum(root)

return self.maxSum

def \_maxPathSum(self, root):

if root is None:

return 0

left = self.\_maxPathSum(root.left)

right = self.\_maxPathSum(root.right)

left = left if left > 0 else 0

right = right if right > 0 else 0

self.maxSum = max(self.maxSum, root.val + left + right)

return max(left, right) + root.val

# 主函数

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

# 树的定义

root = TreeNode(1)

root.left = TreeNode(2)

root.right = TreeNode(3)

solution = Solution()

print("路径和最大为：", solution.maxPathSum(root))

### 【例158】验证二叉树

class TreeNode:

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

self.val = val

self.left, self.right = None, None

class Solution:

def isValidBST(self, root):

self.lastVal = None

self.isBST = True

self.validate(root)

return self.isBST

def validate(self, root):

if root is None:

return

self.validate(root.left)

if self.lastVal is not None and self.lastVal >= root.val:

self.isBST = False

return

self.lastVal = root.val

self.validate(root.right)

# 主函数

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

root = TreeNode(2)

root.left = TreeNode(1)

root.right = TreeNode(4)

root.right.left = TreeNode(3)

root.right.right = TreeNode(5)

solution = Solution()

print("是否是BST：", solution.isValidBST(root))

### 【例159】二叉树的最大深度

class TreeNode:

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

self.val = val

self.left, self.right = None, None

class Solution:

def maxDepth(self, root):

if root is None:

return 0

return max(self.maxDepth(root.left), self.maxDepth(root.right)) + 1

# 主函数

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

root = TreeNode(1)

root.left = TreeNode(2)

root.right = TreeNode(3)

root.right.left = TreeNode(4)

root.right.right = TreeNode(5)

solution = Solution()

print("树的最大深度是：", solution.maxDepth(root))

### 【例160】二叉树的前序遍历

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ArrayList中包含节点值的前序遍历

def preorderTraversal(self, root):

self.results = []

self.traverse(root)

return self.results

def traverse(self, root):

if root is None:

return

self.results.append(root.val)

self.traverse(root.left)

self.traverse(root.right)

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.right = TreeNode(2)

root.right.left = TreeNode(3)

print("要遍历的树是：", printTree(root))

solution = Solution()

print("前序遍历的结果是：", solution.preorderTraversal(root))

### 【例161】二叉树的中序遍历

#### 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ArrayList中包含节点值的前序遍历

def inorderTraversal(self, root):

if root is None:

return []

return self.inorderTraversal(root.left) + [root.val] \

+ self.inorderTraversal(root.right)

# 主函数

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

root = TreeNode(1)

root.right = TreeNode(2)

root.right.left = TreeNode(3)

solution = Solution()

print("树的中序遍历是：", solution.inorderTraversal(root))

### 【例162】将排序列表转换成二叉搜索树

#### 3.代码实现

# 链表的定义

class ListNode(object):

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

self.val = val

self.next = next

# 树的定义

class TreeNode:

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

self.val = val

self.left, self.right = None, None

class Solution:

#参数head是连接链表的第一个节点

#返回一个树节点

def sortedListToBST(self, head):

if not head:

return head

if not head.next:

return TreeNode(head.val)

slow, fast = head, head.next

while fast.next and fast.next.next:

slow = slow.next

fast = fast.next.next

mid = slow.next

slow.next = None

root = TreeNode(mid.val)

root.left = self.sortedListToBST(head)

root.right = self.sortedListToBST(mid.next)

return root

# 层次打印树函数

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\_\_':

head = ListNode(1)

head.next = ListNode(2)

head.next.next = ListNode(3)

solution = Solution()

print("转换后的二叉搜索树是：", printTree(solution.sortedListToBST(head)))

### 【例163】二叉树的最小深度

#### 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

#返回值是一个整数

def minDepth(self, root):

return self.find(root)

def find(self, node):

if node is None:

return 0

left, right = 0, 0

if node.left != None:

left = self.find(node.left)

else:

return self.find(node.right) + 1

if node.right != None:

right = self.find(node.right)

else:

return left + 1

return min(left, right) + 1

# 主函数

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

root = TreeNode(1)

root.left = TreeNode(2)

root.right = TreeNode(3)

root.right.left = TreeNode(4)

root.right.right = TreeNode(5)

solution = Solution()

print("二叉树的最小深度是：", solution.minDepth(root))

### 【例164】不同的二叉搜索树

#### 3.代码实现

class TreeNode:

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

self.val = val

self.left, self.right = None, None

class Solution:

def generateTrees(self, n):

return self.dfs(1, n)

def dfs(self, start, end):

if start > end: return [None]

res = []

for rootval in range(start, end + 1):

LeftTree = self.dfs(start, rootval - 1)

RightTree = self.dfs(rootval + 1, end)

for i in LeftTree:

for j in RightTree:

root = TreeNode(rootval)

root.left = i

root.right = j

res.append(root)

return res

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\_\_':

n = int(input("请输入一个整数："))

solution = Solution()

list = solution.generateTrees(n)

print("生成树的层次遍历分别是：")

for i in list:

print(printTree(i))

### 【例165】将二叉树拆成链表

#### 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

#返回结果值

def flatten(self, root):

self.helper(root)

#重组并按顺序返回最后一个节点

def helper(self, root):

if root is None:

return None

left\_last = self.helper(root.left)

right\_last = self.helper(root.right)

#连接

if left\_last is not None:

left\_last.right = root.right

root.right = root.left

root.left = None

if right\_last is not None:

return right\_last

if left\_last is not None:

return left\_last

return root

#打印树函数

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(2)

root.right = TreeNode(5)

root.left.left = TreeNode(3)

root.left.right = TreeNode(4)

root.right.right = TreeNode(6)

solution = Solution()

solution.flatten(root)

print("变形后的结果是：", printTree(root))

### 【例166】排序数组转为高度最小二叉搜索树

#### 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A是一个整数数组

#返回值是一个树节点

def sortedArrayToBST(self, A):

return self.convert(A, 0, len(A) - 1)

def convert(self, A, start, end):

if start > end:

return None

if start == end:

return TreeNode(A[start])

mid = (start + end) // 2

root = TreeNode(A[mid])

root.left = self.convert(A, start, mid - 1)

root.right = self.convert(A, mid + 1, end)

return root

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\_\_':

list = [1, 2, 3, 4, 5, 6, 7]

print("列表为：", list)

solution = Solution()

print("排列完成的二叉树是：", printTree(solution.sortedArrayToBST(list)))

### 【例167】最近二叉搜索树值Ⅰ

# 树的定义  
class TreeNode:  
 def \_\_init\_\_(self, val):  
 self.val = val  
 self.left, self.right = None, None  
class Solution:  
#参数root是一个给定的二叉搜索树  
#参数target是一个给定的目标值  
#返回值是二叉树中最接近target的值  
 def closestValue(self, root, target):  
 upper = root  
 lower = root  
 while root:  
 if root.val > target:  
 upper = root  
 root = root.left  
 elif root.val < target:  
 lower = root  
 root = root.right  
 else:  
 return root.val  
 if abs(upper.val - target) > abs(lower.val - target):  
 return lower.val  
 return upper.val  
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)  
 print(res)  
if \_\_name\_\_ == '\_\_main\_\_':  
 root = TreeNode(1)  
 solution = Solution()  
 print("原始二叉树为")  
 printTree(root)  
 target = 4.428571  
 print("target=", target)  
 root0 = solution.closestValue(root, target)  
 print("最接近的target是：", root0)

### 【例168】最近二叉搜索树值Ⅱ

#### 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是给定的二叉搜索树  
#参数target是给定的目标值  
#参数k是一个给定的k个值  
#返回值是在二叉搜索树中最接近target的k个值  
 def closestKValues(self, root, target, k):  
 stack\_upper = []  
 stack\_lower = []  
 cur = root  
 while cur:  
 stack\_upper.append(cur)  
 cur = cur.left  
 cur = root  
 while cur:  
 stack\_lower.append(cur)  
 cur = cur.right  
 while len(stack\_upper) > 0 and stack\_upper[-1].val < target:  
 self.move\_upper(stack\_upper)  
 while len(stack\_lower) > 0 and stack\_lower[-1].val >= target:  
 self.move\_lower(stack\_lower)  
 ans = []  
 for i in range(k):  
 if len(stack\_lower) == 0:  
 upper = stack\_upper[-1].val  
 ans.append(upper)  
 self.move\_upper(stack\_upper)  
 elif len(stack\_upper) == 0:  
 lower = stack\_lower[-1].val  
 ans.append(lower)  
 self.move\_lower(stack\_lower)  
 else:  
 upper, lower = stack\_upper[-1].val, stack\_lower[-1].val  
 if upper - target < target - lower:  
 ans.append(upper)  
 self.move\_upper(stack\_upper)  
 else:  
 ans.append(lower)  
 self.move\_lower(stack\_lower)  
 return ans  
 def move\_upper(self, stack):  
 cur = stack.pop()  
 if cur.right:  
 cur = cur.right  
 while cur:  
 stack.append(cur)  
 cur = cur.left  
 def move\_lower(self, stack):  
 cur = stack.pop()  
 if cur.left:  
 cur = cur.left  
 while cur:  
 stack.append(cur)  
 cur = cur.right  
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)  
 print(res)  
if \_\_name\_\_ == '\_\_main\_\_':  
 root = TreeNode(1)  
 solution = Solution()  
 print("原始二叉树为")  
 printTree(root)  
 target = 0.000000  
 k = 1  
 print("target=", target, "\nk=", k)  
 root0 = solution.closestKValues(root, target, k)  
 print("最接近的target是：", root0)

### 【例169】买卖股票的最佳时机Ⅰ

import sys

class Solution:

def maxProfit(self, prices):

total = 0

low, high = sys.maxsize, 0

for x in prices:

if x - low > total:

total = x - low

if x < low:

low = x

return total

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

temp = Solution()

nums1 = [3,2,3,1,2]

nums2 = [5,3,3,4,6]

print(("输入："+str(nums1)))

print(("输出："+str(temp.maxProfit(nums1))))

print(("输入："+str(nums2)))

print(("输出："+str(temp.maxProfit(nums2))))

### 【例170】买卖股票的最佳时机Ⅱ

import sys

class Solution:

def maxProfit(self, prices):

total = 0

low, high = sys.maxsize, sys.maxsize

for x in prices:

if x > high:

high = x

else:

total += high - low

high, low = x, x

return total + high - low

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

temp = Solution()

nums1 = [2,1,2,0,1]

nums2 = [5,3,3,4,6]

print(("输入："+str(nums1)))

print(("输出："+str(temp.maxProfit(nums1))))

print(("输入："+str(nums2)))

print(("输出："+str(temp.maxProfit(nums2))))

### 【例171】买卖股票的最佳时机Ⅲ

class Solution:

def maxProfit(self, prices):

n = len(prices)

if n <= 1:

return 0

p1 = [0] \* n

p2 = [0] \* n

minV = prices[0]

for i in range(1,n):

minV = min(minV, prices[i])

p1[i] = max(p1[i - 1], prices[i] - minV)

maxV = prices[-1]

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

maxV = max(maxV, prices[i])

p2[i] = max(p2[i + 1], maxV - prices[i])

res = 0

for i in range(n):

res = max(res, p1[i] + p2[i])

return res

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

temp = Solution()

nums1 = [2,1,3,5,5,3,2,1]

nums2 = [1,3,1,4,4,2,5,3]

print ("输入："+str(nums1))

print ("输出："+str(temp.maxProfit(nums1)))

print ("输入："+str(nums2))

print ("输出："+str(temp.maxProfit(nums2)))

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 ("输入的数组："+"[2,2,2,3,3,3,3]"+"\n输出："+str(temp.majorityNumber(nums1)))

print ("输入的数组："+"[1,2,3,4]"+"\n输出："+str(temp.majorityNumber(nums2)))

### 【例173】主元素Ⅱ

class Solution:  
 def majorityNumber(self, nums, k):  
 counts = {}  
 max = 0  
 for num in nums:  
 counts[num] = counts.get(num, 0) + 1  
 if counts[num] > max:  
 max = counts[num]  
 majority = num  
 return majority  
# 主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 nums = [3, 1, 2, 3, 2, 3, 3, 4, 4, 4]  
 k = 3  
 # 创建对象  
 solution = Solution()  
 print("输入的数组是： ", nums)  
 print("输出的结果是：", solution.majorityNumber(nums, k))

### 【例174】第K大元素

class Solution:

def kthLargestElement(self, k, A):

if not A or k < 1 or k > len(A):

return None

return self.partition(A, 0, len(A) - 1, len(A) - k)

def partition(self, nums, start, end, k):

if start == end:

return nums[k]

left, right = start, end

pivot = nums[(start + end) // 2]

while left <= right:

while left <= right and nums[left] < pivot:

left += 1

while left <= right and nums[right] > pivot:

right -= 1

if left <= right:

nums[left], nums[right] = nums[right], nums[left]

left, right = left + 1, right - 1

# left is not bigger than right

if k <= right:

return self.partition(nums, start, right, k)

if k >= left:

return self.partition(nums, left, end, k)

return nums[k]

# 主函数

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

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

k = 2

print('初始数组和k值：', A, k)

solution = Solution()

print('第{}大元素是:'.format(k), solution.kthLargestElement(k, A))

### 【例175】滑动窗口内唯一元素数量和

#### 3.代码实现

from collections import Counter  
class Solution:  
 count, keylist = 0, []  
 def Add(self, value):  
 self.count += 1  
 self.keylist.append(value)  
 def Remove(self, value):  
 self.count -= 1  
 self.keylist.remove(value)  
 def slidingWindowUniqueElementsSum(self, nums, k):  
 res = 0  
 if len(nums) <= k:  
 d = Counter(nums)  
 for key in d:  
 if d[key] == 1:  
 res += 1  
 else:  
 dic = Counter(nums[:k])  
 for key in dic:  
 if dic[key] == 1:  
 self.Add(key)  
 start, end = 0, k - 1  
 res += self.count  
 while end + 1 < len(nums):  
 v, u = nums[start], nums[end + 1]  
 dic[v] -= 1  
 if dic[v] == 0 and v in self.keylist:  
 del dic[v]  
 self.Remove(v)  
 if u not in dic and u not in self.keylist:  
 dic[u] = 0  
 self.Add(u)  
 dic[u] += 1  
 if dic[u] == 2 and u in self.keylist:  
 self.Remove(u)  
 if v in dic and dic[v] == 1 and v not in self.keylist:  
 self.Add(v)  
 res += self.count  
 start += 1  
 end += 1  
 return res  
# 主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 nums = [1, 2, 1, 3, 3]  
 k = 3  
 # 创建对象  
 solution = Solution()  
 print("输入的数组是nums=：", nums, "滑动窗口的大小k=", k)  
 print("每一个窗口内唯一元素的个数和是:", solution.slidingWindowUniqueElementsSum(nums, k))

### 【例176】单词拆分Ⅰ

class Solution:

def wordBreak(self, s, dict):

if len(dict) == 0:

return len(s) == 0

n = len(s)

f = [False] \* (n + 1)

f[0] = True

maxLength = max([len(w) for w in dict])

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

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

if not f[i - j]:

continue

if s[i - j:i] in dict:

f[i] = True

break

return f[n]

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

temp = Solution()

string1 = "helloworld"

List = ["hello","world","hahah"]

print(("输入："+string1+" "+str(List)))

print(("输出："+str(temp.wordBreak(string1,List))))

### 【例177】单词拆分Ⅱ

class Solution:

def wordBreak(self, s, wordDict):

return self.dfs(s, wordDict, {})

#找到s的所有切割方案并返回

def dfs(self, s, wordDict, memo):

if s in memo:

return memo[s]

if len(s) == 0:

return []

partitions = []

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

prefix = s[:i]

if prefix not in wordDict:

continue

sub\_partitions = self.dfs(s[i:], wordDict, memo)

for partition in sub\_partitions:

partitions.append(prefix + " " + partition)

if s in wordDict:

partitions.append(s)

memo[s] = partitions

return partitions

# 主函数

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

s = "expressions"

wordDict = set(["express", "ex", "press", "demo", "ions"])

print("String 是:", s)

print("dict是:", wordDict)

solution = Solution()

print("结果是：", solution.wordBreak(s, wordDict))

#### 4.运行结果

String 是: expressions

dict是: {'press', 'ex', 'express', 'ions', 'demo'}

结果是： ['ex press ions', 'express ions']

### 【例178】单词矩阵

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() # remove the last word

# 主函数

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

word = ["area", "lead", "wall", "lady", "ball"]

print("单词序列是：", word)

solution = Solution()

print("构成的单词矩阵是：", solution.wordSquares(word))

### 【例179】单词搜索

class Solution:

def exist(self, board, word):

if word == []:

return True

m = len(board)

if m == 0:

return False

n = len(board[0])

if n == 0:

return False

#访问矩阵

visited = [[False for j in range(n)] for i in range(m)]

for i in range(m):

for j in range(n):

if self.exist2(board, word, visited, i, j):

return True

return False

def exist2(self, board, word, visited, row, col):

if word == '':

return True

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

if row < 0 or row >= m or col < 0 or col >= n:

return False

if board[row][col] == word[0] and not visited[row][col]:

visited[row][col] = True

# row - 1, col

if self.exist2(board, word[1:], visited, row - 1, col) or self.exist2(board, word[1:], visited, row,

col - 1) or self.exist2(board,

word[1:],

visited,

row + 1,

col) or self.exist2(

board, word[1:], visited, row, col + 1):

return True

else:

visited[row][col] = False

return False

#主函数

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

board = ["ABCE", "SFCS", "ADEE"]

word1 = "ABCCED"

word2 = "ABCB"

solution = Solution()

print("board是：", board)

print("word1是：", word1, ",结果是：", solution.exist(board, word1))

print("word2是：", word2, ",结果是：", solution.exist(board, word2))

### 【例180】单词接龙Ⅰ

from collections import deque

class Solution:

def findLadders(self, start, end, dict):

dict.add(start)

dict.add(end)

indexes = self.build\_indexes(dict)

distance = {}

self.bfs(end, start, distance, indexes)

results = []

self.dfs(start, end, distance, indexes, [start], results)

return results

def build\_indexes(self, dict):

indexes = {}

for word in dict:

for i in range(len(word)):

key = word[:i] + '%' + word[i + 1:]

if key in indexes:

indexes[key].add(word)

else:

indexes[key] = set([word])

return indexes

def bfs(self, start, end, distance, indexes):

distance[start] = 0

queue = deque([start])

while queue:

word = queue.popleft()

for next\_word in self.get\_next\_words(word, indexes):

if next\_word not in distance:

distance[next\_word] = distance[word] + 1

queue.append(next\_word)

def get\_next\_words(self, word, indexes):

words = []

for i in range(len(word)):

key = word[:i] + '%' + word[i + 1:]

for w in indexes.get(key, []):

words.append(w)

return words

def dfs(self, curt, target, distance, indexes, path, results):

if curt == target:

results.append(list(path))

return

for word in self.get\_next\_words(curt, indexes):

if distance[word] != distance[curt] - 1:

continue

path.append(word)

self.dfs(word, target, distance, indexes, path, results)

path.pop()

#主函数

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

start = "hit"

end = "cog"

dict = set(["hot", "dot", "dog", "lot", "log"])

print("start是：", start)

print("end是：", end)

print("dict是：", dict)

solution = Solution()

print("结果是：", solution.findLadders(start, end, dict))

### 【例181】单词接龙Ⅱ

import collections

class Solution:

def ladderLength(self, start, end, dict):

dict.add(end)

queue = collections.deque([start])

visited = set([start])

distance = 0

while queue:

distance += 1

for i in range(len(queue)):

word = queue.popleft()

if word == end:

return distance

for next\_word in self.get\_next\_words(word):

if next\_word not in dict or next\_word in visited:

continue

queue.append(next\_word)

visited.add(next\_word)

return 0

def get\_next\_words(self, word):

words = []

for i in range(len(word)):

left, right = word[:i], word[i + 1:]

for char in 'abcdefghijklmnopqrstuvwxyz':

if word[i] == char:

continue

words.append(left + char + right)

return words

#主函数

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

start = "hit"

end = "cog"

dict = {"hot", "dot", "dog", "lot", "log"}

print("start是：", start)

print("end是：", end)

print("dict是：", dict)

solution = Solution()

print("它的长度是：", solution.ladderLength(start, end, dict))

### 【例182】包含所有单词连接的子串

#### 3.代码实现

class Solution(object):

def findSubstring(self, s, words):

hash = {}

res = []

wsize = len(words[0])

for str in words:

if str in hash:

hash[str] += 1

else:

hash[str] = 1

for start in range(0, len(words[0])):

slidingWindow = {}

wCount = 0

for i in range(start, len(s), wsize):

word = s[i : i + wsize]

if word in hash:

if word in slidingWindow:

slidingWindow[word] += 1

else:

slidingWindow[word] = 1

wCount += 1

while hash[word] < slidingWindow[word]:

pos = i - wsize \* (wCount - 1)

removeWord = s[pos : pos + wsize]

print(i, removeWord)

slidingWindow[removeWord] -= 1

wCount -= 1

else:

slidingWindow.clear()

wCount = 0

if wCount == len(words):

res.append(i - wsize \* (wCount - 1))

return res

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

temp = Solution()

string1 = "barfoothefoobarman"

List1 = ["foo","bar"]

print(("输入："+str(string1)+" "+str(List1)))

print(("输出："+str(temp.findSubstring(string1,List1))))

### 【例183】最后一个单词的长度

class Solution:

def lengthOfLastWord(self, s):

return len(s.strip().split(' ')[-1])

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

temp = Solution()

string1 = "hello world"

print(("输入："+string1))

print(("输出："+str(temp.lengthOfLastWord(string1))))

### 【例184】电话号码的字母组合

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185】会议室Ⅰ

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186】会议室Ⅱ

class Interval(object):

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

self.start = start

self.end = end

class Solution(object):

def canAttendMeetings(self, intervals):

"""

intervals的类型是列表，返回值的类型是布尔值

"""

if len(intervals) == 0:

return True

intervals = sorted(intervals, key=lambda x: x.start)

end = intervals[0].end

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

if end > intervals[i].start:

return False

end = intervals[i].end

return True

#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("是否参加所有会议：", solution.canAttendMeetings(intervals))

### 【例187】区间最小数

#### 3.代码实现

class Interval(object):  
 def \_\_init\_\_(self, start, end):  
 self.start = start  
 self.end = end  
class SegmentTree(object):  
 def \_\_init\_\_(self, start, end, min=0):  
 self.start = start  
 self.end = end  
 self.min = min  
 self.left, self.right = None, None  
 @classmethod  
 def build(cls, start, end, a):  
 if start > end:  
 return None  
 if start == end:  
 return SegmentTree(start, end, a[start])  
 node = SegmentTree(start, end, a[start])  
 mid = (start + end) // 2  
 node.left = cls.build(start, mid, a)  
 node.right = cls.build(mid + 1, end, a)  
 node.min = min(node.left.min, node.right.min)  
 return node  
 @classmethod  
 def query(self, root, start, end):  
 if root.start > end or root.end < start:  
 return 0x7fffff  
 if start <= root.start and root.end <= end:  
 return root.min  
 return min(self.query(root.left, start, end), \  
 self.query(root.right, start, end))  
class Solution:  
 """  
 参数A、queries是一个给定的整数数组和一个间隔列表，第i个查询是[queries[i-1].start, queries[i-1].end]，返回结果列表  
 """  
 def intervalMinNumber(self, A, queries):  
 root = SegmentTree.build(0, len(A) - 1, A)  
 result = []  
 for query in queries:  
 result.append(SegmentTree.query(root, query.start, query.end))  
 return result  
#主函数  
if \_\_name\_\_ == '\_\_main\_\_':  
 A = [1, 2, 7, 8, 5]  
 print("输入的数组是：", A)  
 interval1 = Interval(1, 2)  
 interval2 = Interval(0, 4)  
 interval3 = Interval(2, 4)  
 print("要查询的区间为：(", interval1.start, ",", interval1.end, "),(", interval2.start, ",", interval2.end, "),(",  
 interval3.start, ",", interval3.end, ")")  
 solution = Solution()  
 print("区间最小数是：", solution.intervalMinNumber(A, [interval1, interval2, interval3]))

### 【例188】搜索区间

class Solution:

def searchRange(self, A, target):

if len(A) == 0:

return [-1, -1]

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

while start + 1 < end:

mid = (start + end) // 2

if A[mid] < target:

start = mid

else:

end = mid

if A[start] == target:

leftBound = start

elif A[end] == target:

leftBound = end

else:

return [-1, -1]

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

while start + 1 < end:

mid = (start + end) // 2

if A[mid] <= target:

start = mid

else:

end = mid

if A[end] == target:

rightBound = end

else:

rightBound = start

return [leftBound, rightBound]

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

temp = Solution()

List1 = [1,2,4,5,6,7,8]

target = 8

print(("输入："+str(List1)+" "+str(target)))

print(("输出："+str(temp.searchRange(List1,target))))

### 【例189】无重叠区间

#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190】区间合并

class Interval(object):

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

self.start = start

self.end = end

class Solution:

def merge(self, intervals):

intervals = sorted(intervals, key=lambda x: x.start)

result = []

for interval in intervals:

if len(result) == 0 or result[-1].end < interval.start:

result.append(interval)

else:

result[-1].end = max(result[-1].end, interval.end)

return result

#主函数

if \_\_name\_\_ == "\_\_main\_\_":

node1 = Interval(1, 3)

node2 = Interval(2, 6)

node3 = Interval(8, 10)

node4 = Interval(15, 18)

list1 = []

# 创建对象

solution = Solution()

print("初始区间：",

[(node1.start, node1.end), (node2.start, node2.end), (node3.start, node3.end), (node4.start, node4.end)])

mind = solution.merge([node1, node2, node3, node4])

for rement in mind:

list1.append((rement.start, rement.end))

print("合并后区间：", list1)

### 【例191】区间求和Ⅰ

#### 3.代码实现

class SegmentTreeNode:  
 def \_\_init\_\_(self, start, end, max):  
 self.start, self.end, self.max = start, end, max  
 self.left, self.right = None, None  
class SegmentTree(object):  
 def \_\_init\_\_(self, start, end, sum=0):  
 self.start = start  
 self.end = end  
 self.sum = sum  
 self.left, self.right = None, None  
 @classmethod  
 def build(cls, start, end, a):  
 if start > end:  
 return None  
 if start == end:  
 return SegmentTree(start, end, a[start])  
 node = SegmentTree(start, end, a[start])  
 mid = (start + end) // 2  
 node.left = cls.build(start, mid, a)  
 node.right = cls.build(mid + 1, end, a)  
 node.sum = node.left.sum + node.right.sum  
 return node  
 @classmethod  
 def modify(cls, root, index, value):  
 if root is None:  
 return  
 if root.start == root.end:  
 root.sum = value  
 return  
 if root.left.end >= index:  
 cls.modify(root.left, index, value)  
 else:  
 cls.modify(root.right, index, value)  
 root.sum = root.left.sum + root.right.sum  
 @classmethod  
 def query(cls, root, start, end):  
 if root.start > end or root.end < start:  
 return 0  
 if start <= root.start and root.end <= end:  
 return root.sum  
 return cls.query(root.left, start, end) + \  
 cls.query(root.right, start, end)  
class Solution:  
 # 参数A是整数序列  
 def \_\_init\_\_(self, A):  
 #   
 self.root = SegmentTree.build(0, len(A) - 1, A)  
 #参数start和end是索引  
 #返回值是从start到end的和  
 def query(self, start, end):  
 return SegmentTree.query(self.root, start, end)  
 #参数index、value是将A[index]修改为value  
 def modify(self, index, value):  
 SegmentTree.modify(self.root, index, value)  
if \_\_name\_\_ == '\_\_main\_\_':  
 A = [1, 2, 7, 8, 5]  
 print("输入的数组是：", A)  
 solution = Solution(A)  
 solution.\_\_init\_\_(A)  
 print("运行query(0,2)：", solution.query(0, 2))  
 print("运行modify(0,4)")  
 solution.modify(0, 4)  
 print("运行query(0,1)：", solution.query(0, 1))  
 print("运行modify(2,1)")  
 solution.modify(2, 1)  
 print("运行query(2,3)：", solution.query(2, 4))

### 【例192】区间求和Ⅱ

#### 3.代码实现

class Interval(object):  
 def \_\_init\_\_(self, start, end):  
 self.start = start  
 self.end = end  
class SegmentTree(object):  
 def \_\_init\_\_(self, start, end, sum=0):  
 self.start = start  
 self.end = end  
 self.sum = sum  
 self.left, self.right = None, None  
 @classmethod  
 def build(cls, start, end, a):  
 if start > end:  
 return None  
 if start == end:  
 return SegmentTree(start, end, a[start])  
 node = SegmentTree(start, end, a[start])  
 mid = (start + end) // 2  
 node.left = cls.build(start, mid, a)  
 node.right = cls.build(mid + 1, end, a)  
 node.sum = node.left.sum + node.right.sum  
 return node  
 @classmethod  
 def query(self, root, start, end):  
 if root.start > end or root.end < start:  
 return 0  
 if start <= root.start and root.end <= end:  
 return root.sum  
 return self.query(root.left, start, end) + \  
 self.query(root.right, start, end)  
class Solution:  
#参数A、queries是给定的一个整数数组和一个区间列表

#第i个查询是[queries[i-1].start, queries[i-1].end]  
#返回结果列表  
 def intervalSum(self, A, queries):  
 root = SegmentTree.build(0, len(A) - 1, A)  
 result = []  
 for query in queries:  
 result.append(SegmentTree.query(root, query.start, query.end))  
 return result  
if \_\_name\_\_ == '\_\_main\_\_':  
 A = [1, 2, 7, 8, 5]  
 print("输入的数组是", A)  
 interval1 = Interval(1, 2)  
 interval2 = Interval(0, 4)  
 interval3 = Interval(2, 4)  
 print("要查询的区间为：(", interval1.start, ",", interval1.end, "),(", interval2.start, ",", interval2.end, "),(",  
 interval3.start, ",", interval3.end, ")")  
 solution = Solution()  
 print("区间求和：", solution.intervalSum(A, [interval1, interval2, interval3]))

### 【例193】是否为子序列

class Solution(object):

def isSubsequence(self, s, t):

i = 0

j = 0

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

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

i += 1

j += 1

if i == len(s):

return True

else :

return False

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

temp = Solution()

string1 = "abc"

string2 = "abcdefg"

print(("输入："+string1+" "+string2))

print(("输出："+str(temp.isSubsequence(string1,string2))))

### 【例194】最长上升子序列

class Solution:

def longestIncreasingSubsequence(self, nums):

if nums is None or not nums:

return 0

dp = [1] \* len(nums)

for curr, val in enumerate(nums):

for prev in range(curr):

if nums[prev] < val:

dp[curr] = max(dp[curr], dp[prev] + 1)

return max(dp)

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

temp = Solution()

List1 = [1,2,4,5,6,7,8]

List2 = [4,2,4,5,3,7]

print(("输入："+str(List1)))

print(("输出："+str(temp.longestIncreasingSubsequence(List1))))

print(("输入："+str(List2)))

print(("输出："+str(temp.longestIncreasingSubsequence(List2))))

### 【例195】有效的括号序列

class Solution(object):  
 def isValidParentheses(self, s):  
 stack = []  
 for ch in s:  
 #压栈  
 if ch == '{' or ch == '[' or ch == '(':  
 stack.append(ch)  
 else:  
 #栈需非空  
 if not stack:  
 return False  
 #判断栈顶是否匹配  
 if ch == ']' and stack[-1] != '[' or ch == ')' and stack[-1] != '(' or ch == '}' and stack[-1] != '{':  
 return False  
 #弹栈  
 stack.pop()  
 return not stack  
#主函数  
if \_\_name\_\_=="\_\_main\_\_":  
 s="([)]"  
 #创建对象  
 solution=Solution()  
 print("输入的包含括号的字符串是：",s)  
 print("输出的结果是：", solution.isValidParentheses(s))

### 【例196】对称树

class TreeNode:

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

self.val = val

self.left, self.right = None, None

class Solution:

def check\_symmetry(self, nodeA, nodeB):

if nodeA is None and nodeB is None:

return True

if nodeA is None or nodeB is None:

return False

if nodeA.val != nodeB.val:

return False

outer\_result = self.check\_symmetry(nodeA.left, nodeB.right)

inner\_result = self.check\_symmetry(nodeA.right, nodeB.left)

return outer\_result and inner\_result

def isSymmetric(self, root):

if root is None:

return True

return self.check\_symmetry(root.left, root.right)

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)

print(res)

#主函数

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

root = TreeNode(1)

root.left = TreeNode(2)

root.right = TreeNode(2)

root.left.left = TreeNode(3)

root.left.right = TreeNode(4)

root.right.left = TreeNode(4)

root.right.right = TreeNode(3)

solution = Solution()

printTree(root)

print("是否是对称的：", solution.isSymmetric(root))

root = TreeNode(1)

root.left = TreeNode(2)

root.right = TreeNode(2)

root.left.right = TreeNode(3)

root.right.right = TreeNode(3)

printTree(root)

print("是否是对称的：", solution.isSymmetric(root))

### 【例197】图是否是树

class Solution:

def validTree(self, n, edges):

if n - 1 != len(edges):

return False

self.father = {i: i for i in range(n)}

self.size = n

for a, b in edges:

self.union(a, b)

return self.size == 1

def union(self, a, b):

root\_a = self.find(a)

root\_b = self.find(b)

if root\_a != root\_b:

self.size -= 1

self.father[root\_a] = root\_b

def find(self, node):

path = []

while node != self.father[node]:

path.append(node)

node = self.father[node]

for n in path:

self.father[n] = node

return node

#主函数

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

n = 5

edges = [[0, 1], [0, 2], [0, 3], [1, 4]]

print("n = ", n)

print("edges是：", edges)

solution = Solution()

print("图是否是树：", solution.validTree(n, edges))

### 【例198】表达树构造

#### 3.代码实现

#参数expression是一个字符串数组

#返回expression树的根节点

class MyTreeNode:

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

self.left = None

self.right = None

self.val = val

self.exp\_node = ExpressionTreeNode(s)

class Solution:

def get\_val(self, a, base):

if a == '+' or a == '-':

if base == sys.maxint:

return base

return 1 + base

if a == '\*' or a == '/':

if base == sys.maxint:

return base

return 2 + base

return sys.maxint

def build(self, expression):

root = self.create\_tree(expression)

return self.copy\_tree(root)

def copy\_tree(self, root):

if not root:

return None

root.exp\_node.left = self.copy\_tree(root.left)

root.exp\_node.right = self.copy\_tree(root.right)

return root.exp\_node

def create\_tree(self, expression):

stack = []

base = 0

for i in range(len(expression)):

if i != len(expression):

if expression[i] == '(':

if base != sys.maxint:

base += 10

continue

elif expression[i] == ')':

if base != sys.maxint:

base -= 10

continue

val = self.get\_val(expression[i], base)

node = MyTreeNode(val, expression[i])

while stack and val <= stack[-1].val:

node.left = stack.pop()

if stack:

stack[-1].right = node

stack.append(node)

if not stack:

return None

return stack[0]

### 【例199】表达式求值

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()  
 # print(after, before, symbol)  
 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("输入的表达式为：", str)  
 print("其表达式对应的数组是：", num)  
 print("表达式的值是：",solution.evaluateExpression(num))

### 【例200】逆波兰表达式求值

class Solution:  
 def evalRPN(self, tokens):  
 stack = []  
 for i in tokens:  
 if i not in ('+', '-', '\*', '/'):  
 stack.append(int(i))  
 else:  
 op2 = stack.pop()  
 op1 = stack.pop()  
 if i == '+': stack.append(op1 + op2)  
 elif i == '-': stack.append(op1 - op2)  
 elif i == '\*': stack.append(op1 \* op2)  
 else: stack.append(int(op1 \* 1.0 / op2))  
 return stack[0]  
#主函数  
if \_\_name\_\_=="\_\_main\_\_":  
 tokens=["2", "1", "+", "3", "\*"]  
 #创建对象  
 solution=Solution()  
 print("输入的逆波兰表达式是：",tokens)  
 print("计算逆波兰表达式的结果是：", solution.evalRPN(tokens))

### 【例201】将表达式转换为逆波兰表达式

#### 3.代码实现

#一个字符串数组  
#该表达式的逆波兰表达式  
class Stack:  
 def \_\_init\_\_(self):  
 self.items = []  
 def isEmpty(self):  
 return len(self.items) == 0  
 def push(self, item):  
 self.items.append(item)  
 def pop(self):  
 return self.items.pop()  
 def peek(self):  
 if not self.isEmpty():  
 return self.items[-1]  
 def size(self):  
 return len(self.items)  
class Solution:  
 def getLevel(self, s):  
 if s == "+" or s == "-":  
 return 1  
 if s == "\*" or s == "/":  
 return 2  
 return 0  
 def convertToRPN(self, expression):  
 RPN = []  
 cal = Stack()  
 for s in expression:  
 if s == "(":  
 cal.push(s)  
 elif s == ")":  
 while not cal.isEmpty() and cal.peek() != "(":  
 RPN.append(cal.peek())  
 cal.pop()  
 cal.pop()  
 elif s.isdigit():  
 RPN.append(s)  
 else:  
 if not cal.isEmpty():  
 if cal.peek() != "(":  
 while self.getLevel(cal.peek()) >= self.getLevel(s):  
 RPN.append(cal.peek())  
 cal.pop()  
 if cal.isEmpty():  
 break  
 cal.push(s)  
 while not cal.isEmpty():  
 RPN.append(cal.peek())  
 cal.pop()  
 return RPN  
#主函数  
if \_\_name\_\_=="\_\_main\_\_":  
 str=["3","-","4","+","5"]  
 #创建对象  
 solution=Solution()  
 print("输入的表达式数组是：",str)  
 print("表达式的逆波兰表达式是",solution.convertToRPN(str))

### 【例202】最长公共子序列

class Solution:

def longestCommonSubsequence(self, A, B):

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

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

for i in range(n):

for j in range(m):

f[i + 1][j + 1] = max(f[i][j + 1], f[i + 1][j])

if A[i] == B[j]:

f[i + 1][j + 1] = f[i][j] + 1

return f[n][m]

#主函数

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

A = "ABCD"

B = "EACB"

print("序列A：", A)

print("序列B：", B)

solution = Solution()

print("最长公共子序列长度：", solution.longestCommonSubsequence(A, B))

### 【例203】乘积最大子序列

class Solution:

def maxProduct(self, nums):

if not nums:

return None

global\_max = prev\_max = prev\_min = nums[0]

for num in nums[1:]:

if num > 0:

curt\_max = max(num, prev\_max \* num)

curt\_min = min(num, prev\_min \* num)

else:

curt\_max = max(num, prev\_min \* num)

curt\_min = min(num, prev\_max \* num)

global\_max = max(global\_max, curt\_max)

prev\_max, prev\_min = curt\_max, curt\_min

return global\_max

#主函数

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

nums = [2, 3, -2, 4]

print("初始序列：", nums)

solution = Solution()

print("乘积最大子序列的积：", solution.maxProduct(nums))

### 【例204】最长上升连续子序列

class Solution:

def longestIncreasingContinuousSubsequence(self, A):

if not A:

return 0

longest, incr, desc = 1, 1, 1

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

if A[i] > A[i - 1]:

incr += 1

desc = 1

elif A[i] < A[i - 1]:

incr = 1

desc += 1

else:

incr = 1

desc = 1

longest = max(longest, max(incr, desc))

return longest

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

temp = Solution()

nums1 = [6,5,4,3,2]

nums2 = [2,1,2,1,2,1]

print ("输入："+str(nums1))

print ("输出："+str(temp.longestIncreasingContinuousSubsequence(nums1)))

print ("输入："+str(nums2))

print ("输出："+str(temp.longestIncreasingContinuousSubsequence(nums2)))

### 【例205】序列重构

class Solution:

def sequenceReconstruction(self, org, seqs):

from collections import defaultdict

edges = defaultdict(list)

indegrees = defaultdict(int)

nodes = set()

for seq in seqs:

nodes |= set(seq)

for i in range(len(seq)):

if i == 0:

indegrees[seq[i]] += 0

if i < len(seq) - 1:

edges[seq[i]].append(seq[i + 1])

indegrees[seq[i + 1]] += 1

cur = [k for k in indegrees if indegrees[k] == 0]

res = []

while len(cur) == 1:

cur\_node = cur.pop()

res.append(cur\_node)

for node in edges[cur\_node]:

indegrees[node] -= 1

if indegrees[node] == 0:

cur.append(node)

if len(cur) > 1:

return False

return len(res) == len(nodes) and res == org

#主函数

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

org = [1, 2, 3]

seqs = [[1, 2], [1, 3]]

solution = Solution()

print("org是：", org, "seqs是：", seqs)

print("可否从seqs唯一重构出org：", solution.sequenceReconstruction(org, seqs))

org = [1, 2, 3]

seqs = [[1, 2], [1, 3], [2, 3]]

print("org是：", org, "seqs是：", seqs)

print("可否从seqs唯一重构出org：", solution.sequenceReconstruction(org, seqs))

#### 4.运行结果

org是：[1, 2, 3] seqs是：[[1, 2], [1, 3]]

可否从seqs唯一重构出org：False

org是：[1, 2, 3] seqs是：[[1, 2], [1, 3], [2, 3]]

可否从seqs唯一重构出org：True

### 【例206】不同的子序列

#### 1.问题描述

给出字符串S和字符串T，计算在S中不同子序列T出现的个数。子序列字符串是原始字符串通过删除一些(或零个)产生的一个新字符串，并且对剩下字符的相对位置没有影响。(例如，“ACE”是“ABCDE”的子序列字符串，而“AEC”不是)。

#### 2.问题示例

给出S="rabbbit"，T="rabbit"，返回3。

#### 3.代码实现

#参数S, T为两个字符串

#返回不同子序列的个数

class Solution:

def numDistinct(self, S, T):

dp = [[0 for j in range(len(T) + 1)] for i in range(len(S) + 1)]

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

dp[i][0] = 1

for i in range(len(S)):

for j in range(len(T)):

if S[i] == T[j]:

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

else:

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

return dp[len(S)][len(T)]

#主函数

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

S = "rabbbit"

T = "rabbit"

print("字符串S：", S)

print("字符串T：", T)

solution = Solution()

print("结果：", solution.numDistinct(S, T))

### 【例207】跳跃游戏Ⅰ

class Solution:

def canJump(self, A):

p = 0

ans = 0

for item in A[:-1]:

ans = max(ans, p + item)

if(ans <= p):

return False

p += 1

return True

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

temp = Solution()

List1 = [1, 1, 0, 1]

List2 = [1, 2, 0, 1]

print(("输入："+str(List1)))

print(("输出："+str(temp.canJump(List1))))

print(("输入："+str(str(List2))))

print(("输出："+str(temp.canJump(List2))))

### 【例208】跳跃游戏Ⅱ

class Solution:

def jump(self, A):

p = [0]

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

while(i + A[i] >= len(p) and len(p) < len(A)):

p.append(p[i] + 1)

return p[-1]

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.jump(List1))))

print(("输入："+str(str(List2))))

print(("输出："+str(temp.jump(List2))))

### 【例209】翻转游戏

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210】棒球游戏

class Solution:  
 def calPoints(self, ops):  
 # Time: O(n)  
 # Space: O(n)  
 history = []  
 for op in ops:  
 if op == 'C':  
 history.pop()  
 elif op == 'D':  
 history.append(history[-1] \* 2)  
 elif op == '+':  
 history.append(history[-1] + history[-2])  
 else:  
 history.append(int(op))  
 return sum(history)  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 ops = ["5", "2", "C", "D", "+"]  
 # 创建对象  
 solution = Solution()  
 print("初始字符串数组是：", ops)  
 print("总得分数是：", solution.calPoints(ops))

### 【例211】中位数

class Solution:  
 def median(self, nums):  
 nums.sort()  
 return nums[(len(nums) - 1) // 2]  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 nums = [7, 9, 4, 5]  
 #创建对象  
 solution = Solution()  
 print("输入的未排序的整数数组是：", nums)  
 print("中位数是：", solution.median(nums))

### 【例212】滑动窗口的中位数

#### 3.代码实现

class HashHeap:  
 def \_\_init\_\_(self, desc=False):  
 self.hash = dict()  
 self.heap = []  
 self.desc = desc  
 @property  
 def size(self):  
 return len(self.heap)  
 def push(self, item):  
 self.heap.append(item)  
 self.hash[item] = self.size - 1  
 self.\_sift\_up(self.size - 1)  
 def pop(self):  
 item = self.heap[0]  
 self.remove(item)  
 return item  
 def top(self):  
 return self.heap[0]  
 def remove(self, item):  
 if item not in self.hash:  
 return  
 index = self.hash[item]  
 self.\_swap(index, self.size - 1)  
 del self.hash[item]  
 self.heap.pop()  
 # in case of the removed item is the last item  
 if index < self.size:  
 self.\_sift\_up(index)  
 self.\_sift\_down(index)  
 def \_smaller(self, left, right):  
 return right < left if self.desc else left < right  
 def \_sift\_up(self, index):  
 while index != 0:  
 parent = index // 2  
 if self.\_smaller(self.heap[parent], self.heap[index]):  
 break  
 self.\_swap(parent, index)  
 index = parent  
 def \_sift\_down(self, index):  
 if index is None:  
 return  
 while index \* 2 < self.size:  
 smallest = index  
 left = index \* 2  
 right = index \* 2 + 1  
 if self.\_smaller(self.heap[left], self.heap[smallest]):  
 smallest = left  
 if right < self.size and self.\_smaller(self.heap[right], self.heap[smallest]):  
 smallest = right  
 if smallest == index:  
 break  
 self.\_swap(index, smallest)  
 index = smallest  
 def \_swap(self, i, j):  
 elem1 = self.heap[i]  
 elem2 = self.heap[j]  
 self.heap[i] = elem2  
 self.heap[j] = elem1  
 self.hash[elem1] = j  
 self.hash[elem2] = i  
class Solution:  
 def medianSlidingWindow(self, nums, k):  
 if not nums or len(nums) < k:  
 return []  
 self.maxheap = HashHeap(desc=True)  
 self.minheap = HashHeap()  
 for i in range(0, k - 1):  
 self.add((nums[i], i))  
 medians = []  
 for i in range(k - 1, len(nums)):  
 self.add((nums[i], i))  
 # print(self.maxheap.heap, self.median, self.minheap.heap)  
 medians.append(self.median)  
 self.remove((nums[i - k + 1], i - k + 1))  
 # print(self.maxheap.heap, self.median, self.minheap.heap)  
 return medians  
 def add(self, item):  
 if self.maxheap.size > self.minheap.size:  
 self.minheap.push(item)  
 else:  
 self.maxheap.push(item)  
 if self.maxheap.size == 0 or self.minheap.size == 0:  
 return  
 if self.maxheap.top() > self.minheap.top():  
 self.maxheap.push(self.minheap.pop())  
 self.minheap.push(self.maxheap.pop())  
 def remove(self, item):  
 self.maxheap.remove(item)  
 self.minheap.remove(item)  
 if self.maxheap.size < self.minheap.size:  
 self.maxheap.push(self.minheap.pop())  
 @property  
 def median(self):  
 return self.maxheap.top()[0]  
if \_\_name\_\_ == '\_\_main\_\_':  
 A = [1, 2, 7, 8, 5]  
 print("输入的数组是：", A)  
 solution = Solution()  
 print("滑动窗口的中位数是：", solution.medianSlidingWindow(A, 3))

### 【例213】数据流中位数

import heapq  
class Solution:  
 def medianII(self, nums):  
 self.minheap, self.maxheap = [], []  
 medians = []  
 for num in nums:  
 self.add(num)  
 medians.append(self.median)  
 return medians  
 @property  
 def median(self):  
 return -self.maxheap[0]  
 def add(self, value):  
 if len(self.maxheap) <= len(self.minheap):  
 heapq.heappush(self.maxheap, -value)  
 else:  
 heapq.heappush(self.minheap, value)  
 if len(self.minheap) == 0 or len(self.maxheap) == 0:  
 return  
 if -self.maxheap[0] > self.minheap[0]:  
 heapq.heappush(self.maxheap, -heapq.heappop(self.minheap))  
 heapq.heappush(self.minheap, -heapq.heappop(self.maxheap))  
if \_\_name\_\_ == '\_\_main\_\_':  
 nums1 = [1, 2, 3, 4, 5]

nums2 = [4, 5, 1, 3, 2, 6, 0]

nums3 = [2, 20, 100]  
 solution = Solution()  
 print("持续进入数组的列表是：", nums1)

print("新数组的中位数是：", solution.medianII(nums1))

print("持续进入数组的列表是：", nums2)  
 print("新数组的中位数是：", solution.medianII(nums2))

print("持续进入数组的列表是：", nums3)  
 print("新数组的中位数是：", solution.medianII(nums3))

### 【例214】两个排序数组的中位数

class Solution:

def findMedianSortedArrays(self, A, B):

n = len(A) + len(B)

if n % 2 == 1:

return self.findKth(A, B, n // 2 + 1)

else:

smaller = self.findKth(A, B, n // 2)

bigger = self.findKth(A, B, n // 2 + 1)

return (smaller + bigger) / 2.0

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

if len(A) == 0:

return B[k - 1]

if len(B) == 0:

return A[k - 1]

if k == 1:

return min(A[0], B[0])

a = A[k // 2 - 1] if len(A) >= k // 2 else None

b = B[k // 2 - 1] if len(B) >= k // 2 else None

if b is None or (a is not None and a < b):

return self.findKth(A[k // 2:], B, k - k // 2)

return self.findKth(A, B[k // 2:], k - k // 2)

#主函数

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

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

B = [2, 3, 4, 5]

print('数组A是：', A)

print('数组B是：', B)

solution = Solution()

print('他们的中位数是：', solution.findMedianSortedArrays(A, B))

### 【例215】打劫房屋Ⅰ

class Solution:

def houseRobber(self, A):

result = 0

f, g, f1, g1 = 0, 0, 0, 0

for x in A:

f1 = g + x

g1 = max(f, g)

g, f = g1, f1

return max(f, g)

#主函数

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

A = [3, 8, 4]

print("房屋存放金钱：", A)

solution = Solution()

print("打劫到的最多金钱：", solution.houseRobber(A))

### 【例216】打劫房屋Ⅱ

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

#返回值是今晚去打劫能够得到的最多金额

def houseRobber3(self, root):

rob, not\_rob = self.visit(root)

return max(rob, not\_rob)

def visit(self, root):

if root is None:

return 0, 0

left\_rob, left\_not\_rob = self.visit(root.left)

right\_rob, right\_not\_rob = self.visit(root.right)

rob = root.val + left\_not\_rob + right\_not\_rob

not\_rob = max(left\_rob, left\_not\_rob) + max(right\_rob, right\_not\_rob)

return rob, not\_rob

#主函数

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

root = TreeNode(3)

root.left = TreeNode(2)

root.right = TreeNode(3)

root.left.right = TreeNode(3)

root.right.right = TreeNode(1)

solution = Solution()

print("最多可以抢劫的金钱数是：", solution.houseRobber3(root))

### 【例217】子集Ⅰ

#### 3.代码实现

from functools import reduce

class Solution:

def subsetsWithDup(self, S):

S.sort()

p = [[S[x] for x in range(len(S)) if i >> x & 1] for i in range(2 \*\* len(S))]

func = lambda x, y: x if y in x else x + [y]

p = reduce(func, [[], ] + p)

return list(reversed(p))

#主函数

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

S = [1, 2, 2]

print("S是：", S)

solution = Solution()

print("可能的子集是：", solution.subsetsWithDup(S))

### 【例218】子集Ⅱ

#### 3.代码实现

class Solution:

def search(self, nums, S, index):

if index == len(nums):

self.results.append(list(S))

return

S.append(nums[index])

self.search(nums, S, index + 1)

S.pop()

self.search(nums, S, index + 1)

def subsets(self, nums):

self.results = []

self.search(sorted(nums), [], 0)

return self.results

#主函数

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

nums = [1, 2, 3]

print("整数集合是：", nums)

solution = Solution()

print("包含的所有子集有：", solution.subsets(nums))

### 【例219】迷宫Ⅰ

#### 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("迷宫是：", maze)

print("初始地点是:", start)

print("终点是：", destination)

solution = Solution()

print("是否可以走出迷宫:", solution.hasPath(maze, start, destination))

### 【例220】迷宫Ⅱ

#### 3.代码实现

class Solution(object):

def shortestDistance(self, maze, start, destination):

if maze is None or len(maze) == 0 or len(maze[0]) == 0:

return -1

marked = set()

dist\_to = {}

pq = IndexPriorityQueue()

x, y = start

pq.push((x, y), 0)

while pq.size() != 0:

(x, y), dist = pq.pop()

if x == destination[0] and y == destination[1]:

return dist

self.relaxVertex(maze, marked, pq, x, y, dist)

return -1

def relaxVertex(self, maze, marked, pq, x, y, dist):

marked.add((x, y))

for key, next\_dist in self.nextSpaces(maze, x, y, dist):

if key in marked:

continue

if pq.contains(key):

if pq.getDistance(key) > next\_dist:

pq.change(key, next\_dist)

else:

pq.push(key, next\_dist)

def nextSpaces(self, maze, x, y, dist):

next\_spaces = []

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

for v in vectors:

next\_x, next\_y = x, y

next\_dist = dist

while self.isSpace(maze, next\_x + v[0], next\_y + v[1]):

next\_x, next\_y = next\_x + v[0], next\_y + v[1]

next\_dist += 1

if next\_x != x or next\_y != y:

next\_spaces.append(((next\_x, next\_y), next\_dist))

return next\_spaces

def isSpace(self, maze, x, y):

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

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

class IndexPriorityQueue(object):

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

self.data = []

self.key\_index = {}

def size(self):

return len(self.data)

def contains(self, key):

return key in self.key\_index

def getDistance(self, key):

return self.data[self.key\_index[key]][1]

def push(self, key, val):

self.data.append((key, val))

index = self.size() - 1

self.key\_index[key] = index

self.shiftUp(index)

def pop(self):

self.swap(0, self.size() - 1)

key, val = self.data.pop()

del self.key\_index[key]

self.shiftDown(0)

return key, val

def change(self, key, val):

index = self.key\_index[key]

self.data[index][1] = val

self.shiftUp(index)

self.shiftDown(index)

def shiftUp(self, index):

while index > 0:

parent = (index - 1) // 2

if not self.less(index, parent):

break

self.swap(index, parent)

index = parent

def shiftDown(self, index):

while index \* 2 + 1 < self.size():

left\_child = index \* 2 + 1

right\_child = left\_child + 1

min\_child = left\_child

if right\_child < self.size() and self.less(right\_child, left\_child):

min\_child = right\_child

if not self.less(min\_child, index):

break

self.swap(min\_child, index)

index = min\_child

def less(self, index1, index2):

return self.data[index1][1] < self.data[index2][1]

def swap(self, index1, index2):

self.data[index1], self.data[index2] = self.data[index2], self.data[index1]

self.key\_index[self.data[index1][0]] = index1

self.key\_index[self.data[index2][0]] = index2

#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("迷宫是：", maze)

print("初始地点是:", start)

print("终点是：", destination)

solution = Solution()

print("最少的步数是:", solution.shortestDistance(maze, start, destination))

### 【例221】迷宫Ⅲ

#### 3.代码实现

#参数rooms是一个m×n的二维网格

#返回结果

class Solution:

def wallsAndGates(self, rooms):

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

return rooms

m = len(rooms)

n = len(rooms[0])

import queue

queue = queue.Queue()

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

for i in range(m):

for j in range(n):

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

queue.put((i, j))

while not queue.empty():

x, y = queue.get()

for dx, dy in directions:

new\_x = x + dx

new\_y = y + dy

if new\_x < 0 or new\_x >= m or new\_y < 0 or new\_y >= n or rooms[new\_x][new\_y] < rooms[x][y] + 1:

continue

rooms[new\_x][new\_y] = rooms[x][y] + 1

queue.put((new\_x, new\_y))

return rooms

#主函数

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

INF = 2147483647

matrix = [[INF, -1, 0, INF],

[INF, INF, INF, -1],

[INF, -1, INF, -1],

[0, -1, INF, INF]]

solution = Solution()

print("运行的结果是：", solution.wallsAndGates(matrix))

### 【例222】迷宫Ⅳ

#### 3.代码实现

class Point:

def \_\_init\_\_(self, a=0, b=0):

self.x = a

self.y = b

class Solution:

def portal(self, grid):

n = len(grid)

m = len(grid[0])

import sys

record = [[sys.maxsize for \_ in range(m)] for i in range(n)]

for i in range(0, n):

for j in range(0, m):

if (grid[i][j] == 'S'):

source = Point(i, j)

record[source.x][source.y] = 0

import queue

q = queue.Queue(maxsize=n \* m)

q.put(source)

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

while not q.empty():

head = q.get()

for dx, dy in d:

x, y = head.x + dx, head.y + dy

if 0 <= x < n and 0 <= y < m and grid[x][y] != '#' and \

record[head.x][head.y] + 1 < record[x][y]:

record[x][y] = record[head.x][head.y] + 1

if grid[x][y] == 'E':

return record[x][y]

q.put(Point(x, y))

return -1

#主函数

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

grid = [

['S', 'E', '\*'],

['\*', '\*', '\*'],

['\*', '\*', '\*']

]

print("迷宫是：", grid)

solution = Solution()

print("离开迷宫需要的最少时间是：", solution.portal(grid))

### 【例223】数字组合Ⅰ

class Solution:

def combine(self, n, k):

self.res = []

tmp = []

self.dfs(n, k, 1, 0, tmp)

return self.res

def dfs(self, n, k, m, p, tmp):

if k == p:

self.res.append(tmp[:])

return

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

tmp.append(i)

self.dfs(n, k, i + 1, p + 1, tmp)

tmp.pop()

#主函数

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

n = int(input("请输入n:"))

k = int(input("请输入k:"))

solution = Solution()

print("结果是：", solution.combine(n, k))

### 【例224】数字组合Ⅱ

class Solution:

def combinationSum2(self, candidates, target):

candidates.sort()

self.ans, tmp, use = [], [], [0] \* len(candidates)

self.dfs(candidates, target, 0, 0, tmp, use)

return self.ans

def dfs(self, can, target, p, now, tmp, use):

if now == target:

self.ans.append(tmp[:])

return

for i in range(p, len(can)):

if now + can[i] <= target and (i == 0 or can[i] != can[i - 1] or use[i - 1] == 1):

tmp.append(can[i])

use[i] = 1

self.dfs(can, target, i + 1, now + can[i], tmp, use)

tmp.pop()

use[i] = 0

#主函数

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

candidates = [10, 1, 6, 7, 2, 1, 5]

target = 8

print("候选数字：", candidates)

print("目标数字：", target)

solution = Solution()

print("结果是：", solution.combinationSum2(candidates, target))

### 【例225】数字组合Ⅲ

class Solution:

def combinationSum(self, candidates, target):

candidates = sorted(list(set(candidates)))

results = []

self.dfs(candidates, target, 0, [], results)

return results

def dfs(self, candidates, target, start, combination, results):

if target == 0:

return results.append(list(combination))

for i in range(start, len(candidates)):

if target < candidates[i]:

return

# [2] => [2,2]

combination.append(candidates[i])

self.dfs(candidates, target - candidates[i], i, combination, results)

# [2,2] => [2]

combination.pop()

#主函数

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

candidates = [2, 3, 6, 7]

target = 7

solution = Solution()

print("candidates是：", candidates, ",target是： ", target)

print("结果是：", solution.combinationSum(candidates, target))

### 【例226】摆动排序问题

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("初始数组是：", nums)

solution = Solution()

solution.wiggleSort(nums)

print("结果是：", nums)

### 【例227】多关键字排序

class Solution:

def multiSort(self, array):

array.sort(key=lambda x: (-x[1], x[0]))

return array

#主函数

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

array = [[2, 50], [1, 50], [3, 100], ]

print('初始数组：', array)

solution = Solution()

print('结果：', solution.multiSort(array))

### 【例228】排颜色

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229】颜色分类

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230】简化路径

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("输入的路径是：",path)  
 print("路径简化后的结果：",solution.simplifyPath(path))

### 【例231】不同的路径Ⅰ

class Solution:

def c(self, m, n):

mp = {}

for i in range(m):

for j in range(n):

if (i == 0 or j == 0):

mp[(i, j)] = 1

else:

mp[(i, j)] = mp[(i - 1, j)] + mp[(i, j - 1)]

return mp[(m - 1, n - 1)]

def uniquePaths(self, m, n):

return self.c(m, n)

#主函数

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

m = 3

n = 3

print("网格行:{}和列：{}".format(m, n))

solution = Solution()

print("路径条数：", solution.c(m, n))

### 【例232】不同的路径Ⅱ

#### 3.代码实现

class Solution:

def uniquePathsWithObstacles(self, obstacleGrid):

mp = obstacleGrid

for i in range(len(mp)):

for j in range(len(mp[i])):

if i == 0 and j == 0:

mp[i][j] = 1 - mp[i][j]

elif i == 0:

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

mp[i][j] = 0

else:

mp[i][j] = mp[i][j - 1]

elif j == 0:

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

mp[i][j] = 0

else:

mp[i][j] = mp[i - 1][j]

else:

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

mp[i][j] = 0

else:

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

if mp[-1][-1] > 2147483647:

return -1

else:

return mp[-1][-1]

#主函数

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

obstacleGrid = [

[0, 0, 0],

[0, 1, 0],

[0, 0, 0]

]

print("初始网格：")

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

print(obstacleGrid[i])

solution = Solution()

print("路径条数：", solution.uniquePathsWithObstacles(obstacleGrid))

### 【例233】换硬币

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234】硬币摆放

import math

class Solution:

def arrangeCoins(self, n):

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

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

temp = Solution()

n1 = 5

n2 = 10

print(("输入："+str(n1)))

print("输出："+str(temp.arrangeCoins(n1)))

print(("输入："+str(n2)))

print("输出："+str(temp.arrangeCoins(n2)))

### 【例235】硬币排成线Ⅰ

class Solution:

def firstWillWin(self, n):

return bool(n % 3)

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

temp = Solution()

n1 = 100

n2 = 200

print("输入："+str(n1))

print(("输出："+str(temp.firstWillWin(n1))))

print("输入："+str(n2))

print(("输出："+str(temp.firstWillWin(n2))))

### 【例236】硬币排成线Ⅱ

class Solution:

def firstWillWin(self, values):  
 if not values:  
 return False  
 if len(values) <= 2:  
 return True  
 n = len(values)  
 #动态规划  
 f = [0] \* 3  
 prefix\_sum = [0] \* 3  
 f[(n - 1) % 3] = prefix\_sum[(n - 1) % 3] = values[n - 1]  
 #按从n-1～0的相反顺序遍历值  
 for i in range(n - 2, -1, -1):  
 prefix\_sum[i % 3] = prefix\_sum[(i + 1) % 3] + values[i]  
 f[i % 3] = max(  
 values[i] + prefix\_sum[(i + 1) % 3] - f[(i + 1) % 3],  
 values[i] + values[i + 1] + prefix\_sum[(i + 2) % 3] - f[(i + 2) % 3],  
 )  
 return f[0] > prefix\_sum[0] - f[0]  
#主函数  
if \_\_name\_\_=="\_\_main\_\_":  
 values=[1,2,4]  
 #创建对象  
 solution=Solution()  
 print("输入的数组是：",values)  
 print("第一个玩家赢的情况是:",solution.firstWillWin(values))

### 【例237】搜索插入位置

class Solution:

def searchInsert(self, A, target):

if len(A) == 0:

return 0

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

# first position >= target

while start + 1 < end:

mid = (start + end) // 2

if A[mid] >= target:

end = mid

else:

start = mid

if A[start] >= target:

return start

if A[end] >= target:

return end

return len(A)

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

temp = Solution()

List1 = [1,2,4,5]

target = 3

print(("输入："+str(List1)+" "+str(target)))

print(("输出："+str(temp.searchInsert(List1,target))))

### 【例238】俄罗斯套娃信封

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239】包裹黑色像素点的最小矩形

class Solution(object):

def minArea(self, image, x, y):

m = len(image)

if m == 0:

return 0

n = len(image[0])

if n == 0:

return 0

start = y

end = n - 1

while start < end:

mid = start + (end - start) // 2 + 1

if self.checkColumn(image, mid):

start = mid

else:

end = mid - 1

right = start

start = 0

end = y

while start < end:

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

if self.checkColumn(image, mid):

end = mid

else:

start = mid + 1

left = start

start = x

end = m - 1

while start < end:

mid = start + (end - start) // 2 + 1

if self.checkRow(image, mid):

start = mid

else:

end = mid - 1

down = start

start = 0

end = x

while start < end:

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

if self.checkRow(image, mid):

end = mid

else:

start = mid + 1

up = start

return (right - left + 1) \* (down - up + 1)

def checkColumn(self, image, col):

for i in range(len(image)):

if image[i][col] == '1':

return True

return False

def checkRow(self, image, row):

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

if image[row][j] == '1':

return True

return False

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

temp = Solution()

image = ["1000","1100","0110"]

x = 1

y = 2

print(("输入："+str(image)))

print(("输入："+str(x)+","+str(y)))

print(("输出："+str(temp.minArea(image,x,y))))

### 【例240】薪水调整

class Solution:

def getCap(self, a, target):

a.sort()

l, r = a[0], a[len(a)-1]

while l+1<r:

mid = l + ((r-l)>>1)

ans = self.total(a, mid)

if ans > target:

r = mid

elif ans < target:

l = mid

else:

return mid

if self.total(a, l) == target:

return l

if self.total(a, r) == target:

return r

def total(self, a, mid):

res = 0

for n in a:

res += max(n, mid)

return res

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

temp = Solution()

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

target = 25

print("输入："+str(A))

print("输入："+str(target))

print("输出："+str(temp.getCap(A,target)))

### 【例241】木材加工

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242】判断数独是否合法

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("初始值是：", board)  
 print("结果是：", solution.isValidSudoku(board))

### 【例243】移除多余字符

class Solution:  
 def removeDuplicateLetters(self, s):  
 vis, num = [False] \* 26, [0] \* 26;  
 S, cnt = [0] \* 30, 0  
 for c in s:  
 num[ord(c) - ord('a')] += 1  
 for c in s:  
 id = ord(c) - ord('a')  
 num[id] -= 1  
 if (vis[id]):  
 continue  
 while cnt > 0 and S[cnt - 1] > id and num[S[cnt - 1]] > 0:  
 vis[S[cnt - 1]] = False  
 cnt -= 1  
 S[cnt] = id  
 cnt += 1  
 vis[id] = True  
 ans = ""  
 for i in range(cnt):  
 ans += chr(ord('a') + S[i])  
 return ans  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 s = "bcabc"  
 #创建对象  
 solution = Solution()  
 print("初始字符串是：", s)  
 print("移除多余字符后的结果是：", solution.removeDuplicateLetters(s))

### 【例244】三元式解析器

class Solution:  
 def parseTernary(self, expression):  
 objects = []  
 i = len(expression) - 1  
 while i >= 1:  
 if expression[i] == '?':  
 left, right = objects.pop(-1), objects.pop(-1)  
 objects.append(left if expression[i - 1] == 'T' else right)  
 i -= 1  
 elif expression[i] != ':':  
 objects.append(expression[i])  
 i -= 1  
 return objects[0]  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 expression = "F?1:T?4:5"  
 #创建对象  
 solution = Solution()  
 print("输入的表达式是：", expression)  
 print("表达式的结果是：", solution.parseTernary(expression))

### 【例245】符号串生成器

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是：", generator, "startSymbol是：", startSymbol, "symbolString是：", symbolString)

print("是否可以被生成", solution.canBeGenerated(generator, startSymbol, symbolString))

### 【例246】用栈实现队列

#### 3.代码实现

class Solution:  
 def \_\_init\_\_(self):  
 self.stack1 = []  
 self.stack2 = []  
 def adjust(self):  
 if len(self.stack2) == 0:  
 while len(self.stack1) != 0:  
 self.stack2.append(self.stack1.pop())  
 def push(self, element):  
 self.stack1.append(element)  
 def top(self):  
 self.adjust()  
 return self.stack2[len(self.stack2) - 1]  
 def pop(self):  
 self.adjust()  
 return self.stack2.pop()  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 solution = Solution()  
 list1 = []  
 solution.push(1)  
 list1.append(solution.pop())  
 solution.push(2)  
 solution.push(3)  
 list1.append(solution.top())  
 list1.append(solution.pop())  
 print("输入的顺序为：push(1),pop(),push(2),push(3),top(),pop()")  
 print("输出的结果为：", list1)

### 【例247】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)  
 def move\_top\_to(self, t):  
 t.add(self.disks.pop())  
 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("初始盘子个数是：", n)  
 print("towers[0]:", towers[0].disks, "towers[1]:", towers[1].disks, "towers[2]:", towers[2].disks)

### 【例248】带最小值操作的栈

#### 3.代码实现

class MinStack:  
 def \_\_init\_\_(self):  
 self.stack = []  
 self.min\_stack = []  
 def push(self, number):  
 self.stack.append(number)  
 if not self.min\_stack or number <= self.min\_stack[-1]:  
 self.min\_stack.append(number)  
 def pop(self):  
 number = self.stack.pop()  
 if number == self.min\_stack[-1]:  
 self.min\_stack.pop()  
 return number  
 def min(self):  
 return self.min\_stack[-1]  
#主函数  
if \_\_name\_\_=="\_\_main\_\_":  
 minZ=MinStack()  
 list=[]  
 minZ.push(1)  
 list.append(minZ.pop())  
 minZ.push(2)  
 minZ.push(3)  
 list.append(minZ.min())  
 minZ.push(1)  
 list.append(minZ.min())  
 print("输入的顺序是：push(1),pop(),push(2),push(3),min(),push(1),min()")  
 print("输出的结果是：",list)

### 【例249】恢复旋转排序数组问题

class Solution:  
 def recoverRotatedSortedArray(self, nums):  
 pos = nums.index(min(nums))  
 i = 0  
 while i < pos:  
 nums.append(nums[0])  
 nums.remove(nums[0])  
 i += 1  
 return nums  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 nums = [4, 5, 1, 2, 3]  
 #创建对象  
 solution = Solution()  
 print("输入的整数数组是 ：", nums)  
 print("恢复的数组是:", solution.recoverRotatedSortedArray(nums))

### 【例250】移动零问题

#### 3.代码实现

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("输入的整数数组是 ：", nums)  
 nums=solution.moveZeroes(nums)  
 print("移动零后的数组是:", nums)

### 【例251】丢失的间隔问题

class Solution:  
 def findMissingRanges(self, nums, lower, upper):  
 result = []  
 nums = [lower-1] + nums + [upper+1]  
 for i in range(1, len(nums)):  
 l = nums[i-1]  
 h = nums[i]  
 if h - l >= 2:  
 if h - l == 2:  
 result.append(str(l+1))  
 else:  
 result.append(str(l+1)+"->"+str(h-1))  
 return result  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 nums = [0, 1, 3, 50, 75]  
 lower=0  
 upper=99  
 #创建对象  
 solution = Solution()  
 print("输入的整数数组nums= ：", nums, "lower=",lower, "upper=",upper)  
 print("缺少的范围结果是:", solution.findMissingRanges(nums,lower,upper))

### 【例252】三个数的最大乘积

class Solution(object):  
 def maximumProduct(self, nums):  
 if not nums or len(nums) == 0:  
 return 0  
 nums.sort()  
 res1 = nums[-1] \* nums[-2] \* nums[-3]  
 res2 = nums[0] \* nums[1] \* nums[-1]  
 return max(res1, res2)  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 nums = [1, 2, 3]  
 #创建对象  
 solution = Solution()  
 print("输入的数组是 ：", nums)  
 print("最大的积是:", solution.maximumProduct(nums))

### 【例253】用循环数组来实现队列

class CircularQueue:

def \_\_init\_\_(self, n):  
 self.queue = []  
 self.size = n  
 self.head = 0  
 def isFull(self):  
 return len(self.queue) - self.head == self.size  
 def isEmpty(self):  
 return len(self.queue) - self.head == 0  
 def enqueue(self, element):  
 self.queue.append(element)  
#返回值是队列中弹出的元素  
 def dequeue(self):  
 self.head += 1   
 return self.queue[self.head - 1]  
#主函数  
if \_\_name\_\_=="\_\_main\_\_":  
 #创建对象  
 cir=CircularQueue(5)  
 print("isFull()=>",cir.isFull())  
 print("isEmpty() =>",cir.isEmpty())  
 cir.enqueue(1)  
 print("dequeue() =>",cir.dequeue())

### 【例254】寻找数据错误

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("输入的初始数组是：", nums)  
 print("输出的结果是：", solution.findErrorNums(nums))

### 【例255】数据流中第一个独特数

#### 3.代码实现

class LinkedNode:  
 def \_\_init\_\_(self, val=None, next=None):  
 self.value = val  
 self.next = next  
class DataStream:  
 def \_\_init\_\_(self):  
 # do intialization if necessary  
 self.dic = {}  
 self.head = LinkedNode()  
 self.tail = self.head  
 self.visited = set()  
#参数num是数据流中的下一个数字  
#没有返回值  
 def add(self, num):  
 if num in self.visited:  
 return  
 else:  
 if num in self.dic:  
 self.remove(num)  
 self.visited.add(num)  
 else:  
 self.dic[num] = self.tail # 存的是前一个node的信息  
 node = LinkedNode(num)  
 self.tail.next = node  
 self.tail = node  
 #返回数据流中第一个独特的数字  
 def firstUnique(self):  
 # print(self.dic)  
 # print(self.head.next.next.value)  
 if self.head.next != None:  
 return self.head.next.value  
 return -1  
 def remove(self, num):  
 prev = self.dic[num]  
 prev.next = prev.next.next  
 del self.dic[num]  
 #改变dic中对应的信息  
 if prev.next != None:  
 self.dic[prev.next.value] = prev  
 else:  
 self.tail = prev  
#主函数  
if \_\_name\_\_=="\_\_main\_\_":  
 list1=[]  
 solution=DataStream()  
 solution.add(1)  
 solution.add(2)  
 list1.append(solution.firstUnique())  
 solution.add(1)  
 list1.append(solution.firstUnique())  
 print("输入的内容分别是：add(1),add(2),firstUnique(),add(1),firstUnique()")  
 print("最终得到的结果是：",list1)

### 【例256】数据流中第一个唯一的数字

#### 3.代码实现

class Solution:  
 def firstUniqueNumber(self, nums, number):  
 if not nums:  
 return -1  
 num\_cnt = {}  
 ans = None  
 for n in nums:  
 num\_cnt[n] = num\_cnt.get(n, 0) + 1  
 if n == number:  
 break  
 for k, v in num\_cnt.items():  
 if v == 1:  
 ans = k  
 break  
 if ans is None or number not in num\_cnt:  
 return -1  
 return k  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 nums = [1, 2, 2, 1, 3, 4, 4, 5, 6]  
 number = 5  
 #创建对象  
 solution = Solution()  
 print("初始化的数组是：", nums, "给定的终止数字是：", number)  
 print("终止数字到达时的第一个唯一数字是：", solution.firstUniqueNumber(nums, number))

### 【例257】二进制中有多少个1

class Solution:

def countOnes(self, num):

total = 0

for i in range(32):

total += num & 1

num >>= 1

return total

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

temp = Solution()

nums1 = 32

nums2 = 15

print(("输入："+str(nums1)))

print(("输出："+str(temp.countOnes(nums1))))

print(("输入："+str(nums2)))

print(("输出："+str(temp.countOnes(nums2))))

### 【例258】找到映射序列

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("输入的两个列表是A= ", A, "B=", B)  
 print("输出的结果是：", solution.anagramMappings(A, B))

### 【例259】旋转图像

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("输入的数组是：", arr)  
 print("旋转后的矩阵是：", solution.rotate(arr))

### 【例260】相反的顺序存储

class ListNode(object):  
 def \_\_init\_\_(self, val, next=None):  
 self.val = val  
 self.next = next  
class Solution:  
 def reverseStore(self, head):  
 ans = []  
 self.helper(head, ans)  
 return ans  
 def helper(self, head, ans):  
 if head is None:  
 return  
 else:  
 self.helper(head.next, ans)  
 ans.append(head.val)  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 node1 = ListNode(1)  
 node2 = ListNode(2)  
 node3 = ListNode(3)  
 node1.next = node2  
 node2.next = node3  
 list1 = []  
 #创建对象  
 solution = Solution()  
 print("初始链表是：", [node1.val, node2.val, node3.val])  
 print("倒序存储到数组中的结果是：", solution.reverseStore(node1))

### 【例261】太平洋和大西洋的水流

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("给定矩阵是：", matrix)

print("满足条件的点坐标是：", solution.pacificAtlantic(matrix))

### 【例262】不同岛屿的个数

#### 3.代码实现

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

from collections import deque

class Solution:

def numberofDistinctIslands(self, grid):

#grid的类型是整数数组

#返回值的类型是整数型

if not grid:

return 0

queue, check, ans = deque(), set(), 0

for i in range(len(grid)):

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

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

path = " "

queue.append((i, j))

grid[i][j] = 0

while queue:

x, y = queue.popleft()

for dx, dy in DIRECTIONS:

new\_x, new\_y = x + dx, y + dy

if self.is\_valid(grid, new\_x, new\_y):

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

grid[new\_x][new\_y] = 0

path += str(new\_x - i) + str(new\_y - j)

if path not in check:

ans += 1

check.add(path)

return ans

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

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

return x >= 0 and x < row and y >= 0 and y < col and grid[x][y] == 1

#主函数

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

grid = [

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

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

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

[0, 0, 0, 1, 1]

]

print("矩阵是：", grid)

solution = Solution()

print("不同岛屿个数是：", solution.numberofDistinctIslands(grid))

### 【例263】岛的周长问题

class Solution:  
 def islandPerimeter(self, grid):  
 if not grid:  
 return 0  
 m = len(grid)  
 n = len(grid[0])  
 result = 0  
 for i in range(m):  
 for j in range(n):  
 if grid[i][j] == 1:  
 result += self.checkSingleIsland(i, j, grid)  
 return result  
 def checkSingleIsland(self, i, j, grid):  
 top = 1 - grid[i - 1][j] if i - 1 >= 0 else 1  
 bottom = 1 - grid[i + 1][j] if i + 1 < len(grid) else 1  
 left = 1 - grid[i][j - 1] if j - 1 >= 0 else 1  
 right = 1 - grid[i][j + 1] if j + 1 < len(grid[0]) else 1  
 return top + bottom + left + right  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 grid = [[0, 1, 0, 0], [1, 1, 1, 0], [0, 1, 0, 0], [1, 1, 0, 0]]  
 #创建对象  
 solution = Solution()  
 print("初始化的数组", grid)  
 print("岛的周长是：", solution.islandPerimeter(grid))

### 【例264】数字三角形

class Solution:

def minimumTotal(self, triangle):

res = [triangle[0]]

N = len(triangle)

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

res.append([])

for j in range(len(triangle[i])):

if j - 1 >= 0 and j < len(triangle[i - 1]):

res[i].append(min(res[i - 1][j - 1], res[i - 1][j]) + triangle[i][j])

elif j - 1 >= 0:

res[i].append(res[i - 1][j - 1] + triangle[i][j])

else:

res[i].append(res[i - 1][j] + triangle[i][j])

minvalue = min(res[N - 1])

return minvalue

#主函数

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

triangle = [

[2],

[3, 4],

[6, 5, 7],

[4, 1, 8, 3]

]

print("初始数字三角形：")

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

print(triangle[i])

solution = Solution()

print("最小路径：", solution.minimumTotal(triangle))

### 【例265】最大正方形

#### 3.代码实现

class Solution:

def maxSquare(self, matrix):

if not matrix or not matrix[0]:

return 0

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

#初始化

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

for i in range(m):

f[0][i] = matrix[0][i]

edge = max(matrix[0])

for i in range(1, n):

f[i][0] = matrix[i][0]

for j in range(1, m):

if matrix[i][j]:

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

else:

f[i][j] = 0

edge = max(edge, max(f[i]))

return edge \* edge

#主函数

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

s = [[1, 0, 1, 0, 0], [1, 0, 1, 1, 1], [1, 1, 1, 1, 1], [1, 0, 0, 1, 0]]

print("初始矩阵：")

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

print(s[i])

solution = Solution()

print("结果：", solution.maxSquare(s))

### 【例266】最大关联集合

#### 3.代码实现

class Solution:  
 def maximumAssociationSet(self, ListA, ListB):  
 fa = list(range(0, 5009))  
 cnt = [1] \* 5009  
 strlist = [""]  
 def gf(u):  
 if fa[u] != u:  
 fa[u] = gf(fa[u])  
 return fa[u]  
 dict = {}  
 tot = 0  
 for i in range(0, len(ListA)):  
 a, b = 0, 0  
 if ListA[i] not in dict:  
 tot += 1  
 dict[ListA[i]] = tot  
 strlist.append(ListA[i])  
 a = dict[ListA[i]]  
 if ListB[i] not in dict:  
 tot += 1  
 dict[ListB[i]] = tot  
 strlist.append(ListB[i])  
 b = dict[ListB[i]]  
 x, y = gf(a), gf(b)  
 if x != y:  
 fa[y] = x  
 cnt[x] += cnt[y]  
 ans = []  
 k, flag = 0, 0  
 for i in range(0, 5000):  
 if k < cnt[gf(i)]:  
 k = cnt[gf(i)]  
 flag = gf(i)  
 for i in range(0, 5000):  
 if gf(i) == flag:  
 ans.append(strlist[i])  
 return ans  
if \_\_name\_\_ == '\_\_main\_\_':  
 ListA = ["abc", "abc", "abc"]  
 ListB = ["bcd", "acd", "def"]  
 print("ListA是：", ListA)  
 print("ListB是：", ListB)  
 solution = Solution()  
 print("最大关联集合是：", solution.maximumAssociationSet(ListA, ListB))

### 【例267】合并K个排序间隔列表

#### 3.代码实现

class Interval(object):  
 def \_\_init\_\_(self, start, end):  
 self.start = start  
 self.end = end  
class Solution:  
 def mergeKSortedIntervalLists(self, intervals):  
 data = []  
 for i in intervals:  
 data += i  
 data.sort(key=lambda t: t.start)  
 res = [data[0]]  
 for d in data:  
 if res[-1].end < d.start:  
 res += [d]  
 else:  
 res[-1].end = max(res[-1].end, d.end)  
 return res  
if \_\_name\_\_ == '\_\_main\_\_':  
 a = Interval(1, 3)  
 b = Interval(4, 7)  
 c = Interval(6, 8)  
 d = Interval(1, 2)  
 e = Interval(9, 10)  
 intervals0 = [[a, b, c], [d, e]]  
 print("K个排序的间隔列表为：\n[")  
 for interval0 in intervals0[0]:  
 print("(", interval0.start, ",", interval0.end, ")")  
 print("]\n[")  
 for interval0 in intervals0[1]:  
 print("(", interval0.start, ",", interval0.end, ")")  
 print("]")  
 solution = Solution()  
 intervals = solution.mergeKSortedIntervalLists(intervals0)  
 print("合并重叠的间隔：")  
 for interval in intervals:  
 print("(", interval.start, ",", interval.end, ")")

### 【例268】合并账户

class Solution:  
 def accountsMerge(self, accounts):  
 self.initialize(len(accounts))  
 email\_to\_ids = self.get\_email\_to\_ids(accounts)  
 for email, ids in email\_to\_ids.items():  
 root\_id = ids[0]  
 for id in ids[1:]:  
 self.union(id, root\_id)  
 id\_to\_email\_set = self.get\_id\_to\_email\_set(accounts)  
 merged\_accounts = []  
 for user\_id, email\_set in id\_to\_email\_set.items():  
 merged\_accounts.append([  
 accounts[user\_id][0],  
 \*sorted(email\_set),  
 ])  
 return merged\_accounts  
 def get\_id\_to\_email\_set(self, accounts):  
 id\_to\_email\_set = {}  
 for user\_id, account in enumerate(accounts):  
 root\_user\_id = self.find(user\_id)  
 email\_set = id\_to\_email\_set.get(root\_user\_id, set())  
 for email in account[1:]:  
 email\_set.add(email)  
 id\_to\_email\_set[root\_user\_id] = email\_set  
 return id\_to\_email\_set  
 def get\_email\_to\_ids(self, accounts):  
 email\_to\_ids = {}  
 for i, account in enumerate(accounts):  
 for email in account[1:]:  
 email\_to\_ids[email] = email\_to\_ids.get(email, [])  
 email\_to\_ids[email].append(i)  
 return email\_to\_ids  
 def initialize(self, n):  
 self.father = {}  
 for i in range(n):  
 self.father[i] = i  
 def union(self, id1, id2):  
 self.father[self.find(id1)] = self.find(id2)  
 def find(self, user\_id):  
 path = []  
 while user\_id != self.father[user\_id]:  
 path.append(user\_id)  
 user\_id = self.father[user\_id]  
 for u in path:  
 self.father[u] = user\_id  
 return user\_id  
if \_\_name\_\_ == '\_\_main\_\_':  
 accounts1 = [["John", "johnsmith@mail.com", "john00@mail.com"],  
 ["John", "johnnybravo@mail.com"],  
 ["John", "johnsmith@mail.com", "john\_newyork@mail.com"],  
 ["Mary", "mary@mail.com"]]  
 solution = Solution()  
 print("合并前的账户是：", accounts1)  
 print("合并后的账户是：", solution.accountsMerge(accounts1))  
 accounts2 = [['Mary', 'mary@mail.com'],  
 ['John', 'johnnybravo@mail.com'],  
 ['John', 'john00@mail.com', 'john\_newyork@mail.com', 'johnsmith@mail.com']]  
 print("合并前的账户是：", accounts2)  
 print("合并后的账户是：", solution.accountsMerge(accounts2))

### 【例269】集合合并

#### 3.代码实现

class Solution:  
 def find(self, x, f):  
 if x != f[x]:  
 f[x] = self.find(f[x], f)  
 return f[x]  
 def setUnion(self, sets):  
 f = {}  
 for s in sets:  
 first = s[0]  
 for x in s:  
 if not x in f:  
 f[x] = first  
 else:  
 fFirst = self.find(first, f)  
 fx = self.find(x, f)  
 if fx != fFirst:  
 f[fx] = fFirst  
 for s in sets:  
 for x in s:  
 self.find(x, f)  
 hashSet = {}  
 n = 0  
 for val in f.values():  
 if not val in hashSet:  
 n += 1  
 hashSet[val] = val  
 return n  
if \_\_name\_\_ == '\_\_main\_\_':  
 list1 = [[1, 2, 3], [3, 9, 7], [4, 5, 10]]  
 print("list1是：", list1)  
 solution = Solution()  
 print("合并后的集合是：", solution.setUnion(list1))  
 list2 = [[1], [1, 2, 3], [4], [8, 7, 4, 5]]  
 print("lis2t是：", list2)  
 print("合并后的集合是：", solution.setUnion(list2))

### 【例270】快乐数判断

class Solution:  
 def isHappy(self, n):  
 d = {}  
 while True:  
 m = 0  
 while n > 0:  
 m += (n % 10) \*\* 2 # 这是得到n的各位数字，直接进行平方  
 n //= 10 # 是对n进行取整  
 if m in d:  
 return False  
 if m == 1:  
 return True  
 d[m] = m # 如果当前m不等于1，且在字典中不存在，则将其添加到字典中  
 n = m  
#主函数  
if \_\_name\_\_ == "\_\_main\_\_":  
 n = 19  
 #创建对象  
 solution = Solution()  
 print("初始的数字是 ", n)  
 print(" 最终结果是：", solution.isHappy(n))

### 【例271】最多有多少个点在一条直线上

class Point:

def \_\_init\_\_(self, a=0, b=0):

self.x = a

self.y = b

class Solution:

def maxPoints(self, points):

len\_points = len(points)

if len\_points <= 1:

return len\_points

max\_count = 0

for index1 in range(0, len\_points):

p1 = points[index1]

gradients = {}

infinite\_count = 0

duplicate\_count = 0

for index2 in range(index1, len\_points):

p2 = points[index2]

dx = p2.x - p1.x

dy = p2.y - p1.y

if 0 == dx and 0 == dy:

duplicate\_count += 1

if 0 == dx:

infinite\_count += 1

else:

g = float(dy) / dx

gradients[g] = (gradients[g] + 1 if g in gradients else 1)

if infinite\_count > max\_count:

max\_count = infinite\_count

for k, v in gradients.items():

v += duplicate\_count

if v > max\_count:

max\_count = v

return max\_count

#主函数

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

point1 = Point(1, 2)

point2 = Point(3, 6)

point3 = Point(0, 0)

point4 = Point(1, 3)

points = [point1, point2, point3, point4]

print("初始点：", [[point1.x, point1.y], [point2.x, point2.y], [point3.x, point3.y], [point4.x, point4.y]])

solution = Solution()

print("结果：", solution.maxPoints(points))

### 【例272】寻找峰值

class Solution:

def findPeak(self, A):

start, end = 1, len(A) - 2

while start + 1 < end:

mid = (start + end) // 2

if A[mid] < A[mid - 1]:

end = mid

elif A[mid] < A[mid + 1]:

start = mid

else:

end = mid

if A[start] < A[end]:

return end

else:

return start

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

temp = Solution()

List1 = [2,5,3,4,6,7,5]

print(("输入："+str(List1)))

print(("输出："+str(temp.findPeak(List1))))

### 【例273】电灯切换

class Solution:

def flipLights(self, n, m):

if m == 0 or n == 0:

return 1

if n == 1:

return 2

elif n == 2:

if m == 1:

return 3

elif m > 1:

return 4

elif n >= 3:

if m == 1:

return 4

elif m == 2:

return 7

elif m > 2:

return 8

#主函数

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

n = 2

m = 1

print("初始值：n={}，m={}".format(n, m))

solution = Solution()

print("结果：", solution.flipLights(n, m))

### 【例274】第K个质数

#### 3.代码实现

class Solution:

def kthPrime(self, n):

prime = [0] \* 100009;

for i in range(2, n):

if prime[i] == 0:

for j in range(2 \* i, n, i):

prime[j] = 1;

ans = 1;

for i in range(2, n):

if prime[i] == 0:

ans += 1

return ans

#主函数

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

n = 11

print("初始质数：", n)

solution = Solution()

print("结果：第{}个质数".format(solution.kthPrime(n)))

### 【例275】最小调整代价

import sys

class Solution:

def MinAdjustmentCost(self, A, target):

f = [[sys.maxsize for j in range(101)] for i in range(len(A) + 1)]

for i in range(101):

f[0][i] = 0

n = len(A)

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

for j in range(101):

if f[i - 1][j] != sys.maxsize:

for k in range(101):

if abs(j - k) <= target:

f[i][k] = min(f[i][k], f[i - 1][j] + abs(A[i - 1] - k))

ans = f[n][100]

for i in range(101):

if f[n][i] < ans:

ans = f[n][i]

return ans

#主函数

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

A = [1, 4, 2, 3]

target = 1

print("初始数组：", A)

print("相邻两个数的最大值：", target)

solution = Solution()

print("最小调整代价：", solution.MinAdjustmentCost(A, target))

### 【例276】背包问题

class Solution:

def backPack(self, m, A):

n = len(A)

f = [[False] \* (m + 1) for \_ in range(n + 1)]

f[0][0] = True

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

f[i][0] = True

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

if j >= A[i - 1]:

f[i][j] = f[i - 1][j] or f[i - 1][j - A[i - 1]]

else:

f[i][j] = f[i - 1][j]

for i in range(m, -1, -1):

if f[n][i]:

return i

return 0

#主函数

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

m = 11

A = [2, 3, 5, 7]

print("背包大小：", m)

print("每个物品大小：", A)

solution = Solution()

print("最多装满的空间：", solution.backPack(m, A))

### 【例277】爬楼梯

class Solution:

def climbStairs(self, n):

if n == 0:

return 1

if n <= 2:

return n

result = [1, 2]

for i in range(n - 2):

result.append(result[-2] + result[-1])

return result[-1]

#主函数

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

n = 3

print("爬的步数：", n)

solution = Solution()

print("结果：", solution.climbStairs(n))

### 【例278】被围绕的区域

class Solution:

def surroundedRegions(self, board):

if not any(board):

return

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

q = [ij for k in range(max(n, m)) for ij in ((0, k), (n - 1, k), (k, 0), (k, m - 1))]

while q:

i, j = q.pop()

if 0 <= i < n and 0 <= j < m and board[i][j] == 'O':

board[i][j] = 'W'

q += (i, j - 1), (i, j + 1), (i - 1, j), (i + 1, j)

board[:] = [['XO'[c == 'W'] for c in row] for row in board]

#主函数

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

board = [["X", "X", "X", "X"],

["X", "O", "O", "X"],

["X", "X", "O", "X"],

["X", "O", "X", "X"]]

print("board形状是：", board)

solution = Solution()

solution.surroundedRegions(board)

print("修改过后的形状是：", board)

### 【例279】编辑距离

class Solution:

def minDistance(self, word1, word2):

n, m = len(word1), len(word2)

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

for i in range(n + 1):

f[i][0] = i

for j in range(m + 1):

f[0][j] = j

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

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

if word1[i - 1] == word2[j - 1]:

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

# equivalent to f[i][j] = f[i - 1][j - 1]

else:

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

return f[n][m]

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

temp = Solution()

string1 = "hello"

string2 = "world"

print(("输入："+string1+" "+string2))

print(("输出："+str(temp.minDistance(string1,string2))))

### 【例280】最大间距

class Solution:

def maximumGap(self, nums):

if (len(nums)<2): return 0

minNum = -1

maxNum = -1

n = len(nums)

for i in range(n):

minNum = self.min(nums[i], minNum)

maxNum = self.max(nums[i], maxNum)

if maxNum==minNum: return 0

average = (maxNum-minNum) \* 1.0 / (n-1)

if average==0: average += 1

localMin = []

localMax = []

for i in range(n):

localMin.append(-1)

localMax.append(-1)

for i in range(n):

t = int((nums[i]-minNum) / average)

localMin[t] = self.min(localMin[t], nums[i])

localMax[t] = self.max(localMax[t], nums[i])

ans = average

left = 0

right = 1

while left<n-1:

while right<n and localMin[right]==-1: right += 1

if right>=n: break

ans = self.max(ans, localMin[right]-localMax[left])

left = right

right += 1

return ans

def min(self, a, b):

if (a==-1): return b

elif (b==-1): return a

elif (a<b): return a

else: return b

def max(self, a, b):

if (a==-1): return b

elif (b==-1): return a

elif (a>b): return a

else: return b

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

temp = Solution()

List1 = [1, 5, 4, 8]

List2 = [6, 5, 9, 1]

print(("输入："+str(List1)))

print(("输出："+str(temp.maximumGap(List1))))

print(("输入："+str(str(List2))))

print(("输出："+str(temp.maximumGap(List2))))

### 【例281】堆化操作

import heapq  
class Solution:  
 def heapify(self, A):  
 heapq.heapify(A)  
if \_\_name\_\_ == '\_\_main\_\_':  
 A = [3, 2, 1, 4, 5]  
 print("输入的堆数组是：", A)  
 solution = Solution()  
 solution.heapify(A)  
 print("堆化后的数组是：", A)

### 【例282】天际线

#### 3.代码实现

class HashHeap:  
 def \_\_init\_\_(self, desc=False):  
 self.hash = dict()  
 self.heap = []  
 self.desc = desc  
 @property  
 def size(self):  
 return len(self.heap)  
 def push(self, item):  
 self.heap.append(item)  
 self.hash[item] = self.size - 1  
 self.\_sift\_up(self.size - 1)  
 def pop(self):  
 item = self.heap[0]  
 self.remove(item)  
 return item  
 def top(self):  
 return self.heap[0]  
 def remove(self, item):  
 if item not in self.hash:  
 return  
 index = self.hash[item]  
 self.\_swap(index, self.size - 1)  
 del self.hash[item]  
 self.heap.pop()  
 #如果删除的是最后一项  
 if index < self.size:  
 self.\_sift\_up(index)  
 self.\_sift\_down(index)  
 def \_smaller(self, left, right):  
 return right < left if self.desc else left < right  
 def \_sift\_up(self, index):  
 while index != 0:  
 parent = (index - 1) // 2  
 if self.\_smaller(self.heap[parent], self.heap[index]):  
 break  
 self.\_swap(parent, index)  
 index = parent  
 def \_sift\_down(self, index):  
 if index is None:  
 return  
 while index \* 2 + 1 < self.size:  
 smallest = index  
 left = index \* 2 + 1  
 right = index \* 2 + 2  
 if self.\_smaller(self.heap[left], self.heap[smallest]):  
 smallest = left  
 if right < self.size and self.\_smaller(self.heap[right], self.heap[smallest]):  
 smallest = right  
 if smallest == index:  
 break  
 self.\_swap(index, smallest)  
 index = smallest  
 def \_swap(self, i, j):  
 elem1 = self.heap[i]  
 elem2 = self.heap[j]  
 self.heap[i] = elem2  
 self.heap[j] = elem1  
 self.hash[elem1] = j  
 self.hash[elem2] = i  
class Solution:  
#参数buildings是一个整数数组  
#返回值是找到这个buildings的轮廓  
 def buildingOutline(self, buildings):  
 points = []  
 for index, (start, end, height) in enumerate(buildings):  
 points.append((start, height, index, True))  
 points.append((end, height, index, False))  
 points = sorted(points)  
 maxheap = HashHeap(desc=True)  
 intervals = []  
 last\_position = None  
 for position, height, index, is\_start in points:  
 max\_height = maxheap.top()[0] if maxheap.size else 0  
 self.merge\_to(intervals, last\_position, position, max\_height)  
 if is\_start:  
 maxheap.push((height, index))  
 else:  
 maxheap.remove((height, index))  
 last\_position = position  
 return intervals  
 def merge\_to(self, intervals, start, end, height):  
 if start is None or height == 0 or start == end:  
 return  
 if not intervals:  
 intervals.append([start, end, height])  
 return  
 \_, prev\_end, prev\_height = intervals[-1]  
 if prev\_height == height and prev\_end == start:  
 intervals[-1][1] = end  
 return  
 intervals.append([start, end, height])  
if \_\_name\_\_ == '\_\_main\_\_':  
 buildings = [[1, 3, 3],  
 [2, 4, 4],  
 [5, 6, 1]]  
 print("三座大楼分别是：", buildings)  
 solution = Solution()  
 print("外轮廓线是：", solution.buildingOutline(buildings))

### 【例283】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\_\_':

n = int(input("请输入一个非负整数；"))

solution = Solution()

print("格雷编码的结果是：", solution.grayCode(n))

### 【例284】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("地图是：", map)

solution = Solution()

print("能否到达终点", solution.reachEndpoint(map))

### 【例285】恢复IP地址

class Solution:

def restoreIpAddresses(self, s):

def dfs(s, sub, ips, ip):

if sub == 4: # should be 4 parts

if s == '':

ips.append(ip[1:]) # remove first '.'

return

for i in range(1, 4): # the three ifs' order cannot be changed!

if i <= len(s): # if i > len(s), s[:i] will make false!!!!

if int(s[:i]) <= 255:

dfs(s[i:], sub + 1, ips, ip + '.' + s[:i])

if s[0] == '0': break # make sure that res just can be '0.0.0.0' and remove like '00'

ips = []

dfs(s, 0, ips, '')

return ips

#主函数

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

solution = Solution()

S = "25525511135"

print("字符串S是：", S)

print("所有可能的IP地址为：", solution.restoreIpAddresses(S))

### 【例286】斐波纳契数列

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287】最长公共前缀

class Solution:

def longestCommonPrefix(self, strs):

if len(strs) <= 1:

return strs[0] if len(strs) == 1 else ""

end, minl = 0, min([len(s) for s in strs])

while end < minl:

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

if strs[i][end] != strs[i-1][end]:

return strs[0][:end]

end = end + 1

return strs[0][:end]

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

temp = Solution()

nums1 = ["ABCD","ABEF","ACEF"]

nums2 = ["BCD","BEF","BCEF"]

print ("输入的数组："+"['ABCD','ABEF','ACEF']")

print ("输出："+str(temp.longestCommonPrefix(nums1)))

print ("输入的数组："+'["BCD","BEF","BCEF"]')

print ("输出："+str(temp.longestCommonPrefix(nums2)))

### 【例288】解码方法

class Solution:

def numDecodings(self, s):

if s == "" or s[0] == '0':

return 0

dp=[1,1]

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

if 10 <= int(s[i - 2 : i]) <=26 and s[i - 1] != '0':

dp.append(dp[i - 1] + dp[i - 2])

elif int(s[i-2 : i]) == 10 or int(s[i - 2 : i]) == 20:

dp.append(dp[i - 2])

elif s[i-1] != '0':

dp.append(dp[i-1])

else:

return 0

return dp[len(s)]

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

temp = Solution()

string1 = "1"

string2 = "23"

print(("输入："+string1))

print(("输出："+str(temp.numDecodings(string1))))

print(("输入："+string2))

print(("输出："+str(temp.numDecodings(string2))))

### 【例289】吹气球

class Solution:

def maxCoins(self, nums):

if not nums:

return 0

nums = [1, \*nums, 1]

n = len(nums)

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

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

for j in range(i + 2, n):

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

dp[i][j] = max(dp[i][j], dp[i][k] + dp[k][j] + nums[i] \* nums[k] \* nums[j])

return dp[0][n - 1]

#主函数

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

nums = [4, 1, 5, 10]

print("初始数组：", nums)

solution = Solution()

print("最多分数：", solution.maxCoins(nums))

### 【例290】生成括号

class Solution:

def helpler(self, l, r, item, res):

if r < l:

return

if l == 0 and r == 0:

res.append(item)

if l > 0:

self.helpler(l - 1, r, item + '(', res)

if r > 0:

self.helpler(l, r - 1, item + ')', res)

def generateParenthesis(self, n):

if n == 0:

return []

res = []

self.helpler(n, n, '', res)

return res

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

temp = Solution()

nums1 = 1

nums2 = 2

print(("输入："+str(nums1)))

print(("输出："+str(temp.generateParenthesis(nums1))))

print(("输入："+str(nums2)))

print(("输出："+str(temp.generateParenthesis(nums2))))

### 【例291】正则表达式匹配

class Solution:

def isMatch(self, source, pattern):

return self.is\_match\_helper(source, 0, pattern, 0, {})

# source 从i开始的后缀能否匹配上，pattern从j开始的后缀

def is\_match\_helper(self, source, i, pattern, j, memo):

if (i, j) in memo:

return memo[(i, j)]

# source是空

if len(source) == i:

return self.is\_empty(pattern[j:])

if len(pattern) == j:

return False

if j + 1 < len(pattern) and pattern[j + 1] == '\*':

matched = self.is\_match\_char(source[i], pattern[j]) and self.is\_match\_helper(source, i + 1, pattern, j,

self.is\_match\_helper(source, i, pattern, j + 2, memo)

else:

matched = self.is\_match\_char(source[i], pattern[j]) and \

self.is\_match\_helper(source, i + 1, pattern, j + 1, memo)

memo[(i, j)] = matched

return matched

def is\_match\_char(self, s, p):

return s == p or p == '.'

def is\_empty(self, pattern):

if len(pattern) % 2 == 1:

return False

for i in range(len(pattern) // 2):

if pattern[i \* 2 + 1] != '\*':

return False

return True

#主函数

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

solution = Solution()

StringA = "aaa"

StringB = "aa"

print("StringA 是：", StringA, "，StringB 是：", StringB, "，它们是否匹配：", solution.isMatch(StringA,StringB))

StringC = "aab"

StringD = "c\*a\*b"

print("StringC 是：", StringC, "，StringD 是：", StringD, "，它们是否匹配：", solution.isMatch(StringC,StringD))

### 【例292】分割标签

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\_\_':

temp = Solution()

string1 = "ababcbacadefegdehijhklij"

print(("输出："+str(temp.partitionLabels(string1))))

### 【例293】装最多水的容器

class Solution(object):

def maxArea(self, height):

left, right = 0, len(height) - 1

ans = 0

while left < right:

if height[left] < height[right]:

area = height[left] \* (right - left)

left += 1

else:

area = height[right] \* (right - left)

right -= 1

ans = max(ans, area)

return ans

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

temp = Solution()

List1 = [1,2,3]

List2 = [2,5,1,3]

print(("输入："+str(List1)))

print(("输出："+str(temp.maxArea(List1))))

print(("输入："+str(List2)))

print(("输出："+str(temp.maxArea(List2))))

### 【例294】接雨水

class Solution:

def trapRainWater(self, heights):

if not heights:

return 0

left, right = 0, len(heights) - 1

left\_max, right\_max = heights[left], heights[right]

water = 0

while left <= right:

if left\_max < right\_max:

left\_max = max(left\_max, heights[left])

water += left\_max - heights[left]

left += 1

else:

right\_max = max(right\_max, heights[right])

water += right\_max - heights[right]

right -= 1

return water

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

temp = Solution()

List1 = [0,1,0,2,1,0,1,3,2,1,2,1]

print(("输入："+str(List1)))

print(("输出："+str(temp.trapRainWater(List1))))

### 【例295】加油站

class Solution:

def canCompleteCircuit(self, gas, cost):

n = len(gas)

diff = []

for i in range(n): diff.append(gas[i]-cost[i])

for i in range(n): diff.append(gas[i]-cost[i])

if n==1:

if diff[0]>=0: return 0

else: return -1

st = 0

now = 1

tot = diff[0]

while st<n:

while tot<0:

st = now

now += 1

tot = diff[st]

if st>n: return -1

while now!=st+n and tot>=0:

tot += diff[now]

now += 1

if now==st+n and tot>=0: return st

return -1

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

temp = Solution()

List1 = [1, 1, 3, 1]

List2 = [2, 2, 1, 1]

print(("输入："+str(List1)+" "+str(List2)))

print(("输出："+str(temp.canCompleteCircuit(List1,List2))))

### 【例296】分糖果

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297】建立邮局

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("网格是：", grid)

solution = Solution()

print("最近的距离是：", solution.shortestDistance(grid))

### 【例298】寻找最便宜的航行旅途

import heapq

class Solution:

def findCheapestPrice(self, n, flights, src, dst, K):

map = {}

for start, end, cost in flights:

if start not in map:

map[start] = [(cost, end)]

else:

map[start].append((cost, end))

if src not in map:

return -1

hq = []

for cost, next\_stop in map[src]:

heapq.heappush(hq, (cost, next\_stop, 0))

while hq:

cml\_cost, cur\_stop, level = heapq.heappop(hq)

if level > K:

continue

elif cur\_stop == dst:

return cml\_cost

if cur\_stop in map:

for next\_cost, next\_stop in map[cur\_stop]:

heapq.heappush(hq, (cml\_cost + next\_cost, next\_stop, level + 1))

return -1

#主函数

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

n = 3

flights = [[0, 1, 100], [1, 2, 100], [0, 2, 500]]

src = 0

dst = 2

k = 1

print("城市总数 = ", n, "每条线路的价格 = ", flights, "出发站 = ", src, "终点站 = ", dst, "中转站 = ", k)

solution = Solution()

print("航班的价格是：", solution.findCheapestPrice(n, flights, src, dst, k))

### 【例299】utf-8编码检查

#### 3.代码实现

#采用utf-8编码格式

def check(nums, start, size):

for i in range(start + 1, start + size + 1):

if i >= len(nums) or (nums[i] >> 6) != 0b10:

return False

return True

class Solution(object):

def validUtf8(self, nums, start=0):

while start < len(nums):

first = nums[start]

if (first >> 3) == 0b11110 and check(nums, start, 3):

start += 4

elif (first >> 4) == 0b1110 and check(nums, start, 2):

start += 3

elif (first >> 5) == 0b110 and check(nums, start, 1):

start += 2

elif (first >> 7) == 0:

start += 1

else:

return False

return True

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

temp = Solution()

nums1 = [235,140,138]

nums2 = [250,125,125]

print(("输入："+str(nums1)))

print(("输出："+str(temp.validUtf8(nums1))))

print(("输入："+str(nums2)))

print(("输出："+str(temp.validUtf8(nums2))))

### 【例300】哈希函数

class Solution:  
 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\_\_":  
 key = "abcd"  
 size = 100  
 #创建对象  
 solution = Solution()  
 print("输入的字符串是 ", key, "哈希表的大小是:", size)  
 print("输出的结果是：", solution.hashCode(key, size))