# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 2 23:43:24 2019

@author: huyn

"""

#123456789 = K

def target(k):

letter = "123456789"

res = []

def dfs(index,currentVal,k,path):

if index==len(letter):

if currentVal ==k:

res.append("".join(path))

else:

for i in range(index,len(letter)):

val = int(letter[index:i+1])

# -

currentVal-=val

path.append("-")

path.append(letter[index:i+1])

dfs(i+1,currentVal,k,path)

currentVal+=val

path.pop()

path.pop()

# +

currentVal+=val

path.append("+")

path.append(letter[index:i+1])

dfs(i+1,currentVal,k,path)

currentVal-=val

path.pop()

path.pop()

for i in range(len(letter)):

val = int(letter[:i+1])

dfs(i+1,-val,k,["-"+letter[:i+1]])

dfs(i+1,val,k,[letter[:i+1]])

return res

#print (target(100))# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 9 16:51:27 2019

@author: huyn

"""

#3Sum

from typing import List

def threeSum(self, nums: List[int]) -> List[List[int]]:

nums.sort()

res =set()

check = set()

for index,num in enumerate(nums):

if num not in check:

check.add(num)

start,stop = index+1,len(nums)-1

s = -num

while start<stop:

val = nums[start]+nums[stop]

if val==s:

res.add((num,nums[start],nums[stop]))

start+=1

stop-=1

elif val<s:

start+=1

else:

stop-=1

return [list(item) for item in res]# -\*- coding: utf-8 -\*-

"""

Created on Thu Sep 19 11:05:08 2019

@author: huyn

"""

import collections

#721. Accounts Merge

#Given a list accounts, each element accounts[i] is a list of strings, where the first element

#accounts[i][0] is a name, and the rest of the elements are emails representing emails of the account.

#

#Now, we would like to merge these accounts. Two accounts definitely belong to the

#same person if there is some email that is common to both accounts. Note that even if

# two accounts have the same name, they may belong to different people as people could have the same name.

# A person can have any number of accounts initially, but all of their accounts definitely have the same name.

#

#After merging the accounts, return the accounts in the following format: the first

#element of each account is the name, and the rest of the elements are emails in sorted order.

#The accounts themselves can be returned in any order.

def accountsMerge(accounts):

em\_to\_name = {}

graph = collections.defaultdict(set) # create a graph that store infor of emails that are unique

# this graph would help us to do connected components later

for acc in accounts:

name = acc[0]

for email in acc[1:]:

# we dont need an edge between every pair of edge, just add edge between first to every other

# and everyother to first, then later on, if appear an edge to another different email

# we can still join them together

graph[acc[1]].add(email)

graph[email].add(acc[1])

# link the email back to the name

em\_to\_name[email] = name

seen = set()

ans = []

for email in graph:

# we do a connected component search here

if email not in seen:

# we add this email to seen

seen.add(email)

# initiate a stack for our component

stack = [email]

component = []

while stack:

node = stack.pop()

component.append(node)

for nei in graph[node]:

if nei not in seen:

seen.add(nei)

stack.append(nei)

# done w stack means we have connected all possible that link to the email, using the map to name

# to create an entry in answer

ans.append([em\_to\_name[email]] + sorted(component))

return ans

#accounts = [["John", "johnsmith@mail.com", "john00@mail.com"], ["John", "johnnybravo@mail.com"], ["John", "johnsmith@mail.com", "john\_newyork@mail.com"], ["Mary", "mary@mail.com"]]

#print (accountsMerge(accounts))

# -\*- coding: utf-8 -\*-

"""

Created on Thu Oct 3 00:44:38 2019

@author: huyn

"""

#67. Add Binary

def addBinary(self, a: str, b: str) -> str:

remainder = 0

res = []

i,j =len(a)-1,len(b)-1

while i>=0 and j>=0:

valA = int(a[i])

valB = int(b[j])

if valA+valB+remainder==3:

res.append("1")

remainder = 1

elif valA+valB+remainder ==2:

res.append("0")

remainder = 1

else:

res.append(str(valA+valB+remainder))

remainder = 0

i-=1

j-=1

while i>=0:

valA = int(a[i])

if valA+remainder==2:

res.append("0")

remainder = 1

else:

res.append(str(valA+remainder))

remainder = 0

i-=1

while j>=0:

valB = int(b[j])

if valB+remainder==2:

res.append("0")

remainder = 1

else:

res.append(str(valB+remainder))

remainder = 0

j-=1

if remainder:

res.append("1")

return "".join(res[::-1])# -\*- coding: utf-8 -\*-

"""

Created on Sun Sep 22 18:13:39 2019

@author: huyn

"""

import random

#282. Expression Add Operators

#Given a string that contains only digits 0-9 and a target value, return all

#possibilities to add binary operators (not unary) +, -, or \* between the digits so they evaluate to the target value.

def addOperators(num: str, target: int):

arr = []

def dfs(start,path,target,accumulate,prevVal):

# print (path,accumulate)

if start == len(num):

if accumulate==target:

arr.append("".join(path))

elif start<len(num):

# print (path)

for i in range(start,len(num)):

string = num[start:i+1]

# check if string is valid

if str(int(string))==string:

try:

val = int(string)

except:

val = 0

path.append("+")

path.append(string)

dfs(i+1,path,target,accumulate+val,val)

path.pop()

path.pop()

path.append("-")

path.append(string)

dfs(i+1,path,target,accumulate-val,-val)

path.pop()

path.pop()

path.append("\*")

path.append(string)

pre = prevVal\*val

dfs(i+1,path,target,accumulate-prevVal+pre,pre)

path.pop()

path.pop() # pop trhe string

for i in range(len(num)):

string = num[:i+1]

if str(int(string))==string:

dfs(i+1,[string],target,int(string),int(string))

return arr

#num ="12345"

#print (addOperators(num,4))

#target = 6

#num = "123"

#print (addOperators(num,target))

#target = 6

#num = "232"

#print (addOperators(num,target))

#target = 8

#num = "232"

#print (addOperators(num,target))

#target = 0

#num = "00"

#print (addOperators(num,target))

for i in range(1):

num = random.randint(1,10\*\*6)

target = random.randint(1,10)

print (""""{}"\n{}""".format(num,target))# -\*- coding: utf-8 -\*-

"""

Created on Thu Oct 10 00:16:14 2019

@author: Huy Nguyen

"""

def addTwoNumbers(self, l1: ListNode, l2: ListNode) -> ListNode:

def dfs(l1,l2,over):

if l1 and l2:

n1 = l1.val

n2 = l2.val

val =n1+n2+over

newNode = ListNode(val%10)

newNode.next = dfs(l1.next,l2.next,val//10)

return newNode

elif l1:

n1 = l1.val

val =n1+over

newNode = ListNode(val%10)

newNode.next = dfs(l1.next,l2,val//10)

return newNode

elif l2:

n2 = l2.val

val =n2+over

newNode = ListNode(val%10)

newNode.next = dfs(l1,l2.next,val//10)

return newNode

else:

if over==1:

return ListNode(1)

return dfs(l1,l2,0)# -\*- coding: utf-8 -\*-

"""

Created on Sun Sep 22 18:43:09 2019

@author: huyn

"""

#269. Alien Dictionary

#There is a new alien language which uses the latin alphabet. However, the order

# among letters are unknown to you. You receive a list of non-empty words from the dictionary,

# where words are sorted lexicographically by the rules of this new language.

# Derive the order of letters in this language.

def alienOrder(words):

res = []

parents = {} # map the backward edge to check whether a character should be a start node

neighbors = {} # store the forward edge

for word in words:

for letter in word:

if letter not in parents:

parents[letter]=set()

if letter not in neighbors:

neighbors[letter] = set()

#

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

w1 = words[i]

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

w2 = words[j]

for k in range(min(len(w1),len(w2))):

if w1[k]==w2[k]:

continue

elif w1[k]!=w2[k]:

smaller = w1[k]

larger = w2[k]

# set up neighbor that map smaller to larger

# make parents as set so we can remove the parent from each node faster

neighbors[smaller].add(larger)

parents[larger].add(smaller)

# remember to break

break

startNodes = [node for node in parents if len(parents[node])==0]

# we keep pop out startnodes

while startNodes:

node = startNodes.pop()

# add this to res, we know for sure it is equal or less to everything on the left

res.append(node)

# for each neighbor, remove edge from neighbor to node as parent

for neighbor in neighbors[node]:

# remove edge

parents[neighbor].remove(node)

# if there are no more parents of neighbor, add it to start node

if not parents[neighbor]:

startNodes.append(neighbor)

# we check if there are some edges for any key in parents node

for node in parents:

if parents[node]:

return ""

return "".join(res)

#words = ["ab","adc"]

#words = ["z","z"]

#print (alienOrder(words))

# -\*- coding: utf-8 -\*-

"""

Created on Tue Oct 8 15:34:55 2019

@author: huyn

"""

#Amazing Number

#Define amazing number as: its value is less than or equal to its index. Given a

#circular array, find the starting position, such that the total number of amazing numbers in the array is maximized.

#Example 1: 0, 1, 2, 3

#Ouptut: 0. When starting point at position 0, all the elements in the array are equal to

#its index. So all the numbers are amazing number.

#Example 2: 1, 0 , 0

#Output: 1. When starting point at position 1, the array becomes 0, 0, 1. All the elements are amazing number.

#If there are multiple positions, return the smallest one.

#

#should get a solution with time complexity less than O(N^2)

def getAmazingNumberNaive(arr):

size = len(arr)

maxCount = 0

for i in range(size):

starting =i

count = 0

for j in range(size):

# print (starting,j)

if j>=arr[starting%size]:

# print ("index:",j, "num:",arr[starting%size])

count+=1

starting+=1

maxCount= max(maxCount,count)

return maxCount

arr =[0,1,2,3]

print (getAmazingNumberNaive(arr))

#arr=[3,2,1,0]

#

#print (getAmazingNumberNaive(arr))

#arr = [4]

#print (getAmazingNumberNaive(arr))# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 9 16:23:36 2019

@author: huyn

"""

#myAtoi

def myAtoi(string: str) -> int:

numeric = []

d ={'7': 7, '3': 3, '8': 8, '6': 6, '5': 5, '2': 2, '1': 1, '4': 4, '9': 9, '0': 0}

for letter in string:

if letter.isalpha():

break

elif letter ==" " and not numeric:

continue

elif letter ==" " and numeric:

break

elif letter.isnumeric():

numeric.append(letter)

elif (letter =="-" or letter =="+") and not numeric:

if letter =="-" :

numeric.append("-")

else:

numeric.append("+")

else:

break

isNeg= False

if not numeric:

return 0

if numeric[0]=="-" or numeric[0]=="+":

if numeric[0]=="-":

isNeg= True

numeric.pop(0)

# we go to the first numeric that is not 0

while numeric:

if numeric[0]=="0":

numeric.pop(0)

else:

break

val = 0

while numeric:

val=val\*10+d[numeric.pop(0)]

if isNeg:

return max(-2147483648,-val)

return min(val,2147483647)# -\*- coding: utf-8 -\*-

"""

Created on Thu Sep 26 14:44:18 2019

@author: huyn

"""

#Balance the parenthesis in a string

#You are given a string with alphanumeric characters and parentheses. Your goal

#is to return a string with balanced parentheses by removing the fewest characters possible.

#Note that you cannot add anything to the string.

#"()" -> "()"

#"b(a)r)" -> "b(a)r"

#")(" -> ""

#"(((((" -> ""

#")(())(" -> "(())"

#string = ")(())("

def balance\_parens\_naive(string):

stack = []

indices = []

for index,item in enumerate(string):

if item ==")":

if not stack:

indices.append(index)

else:

stack.pop()

elif item =="(":

stack.append(index)

indices.extend(stack)

return "".join([string[i] for i in range(len(string)) if i not in indices])

# O(n) time, O(n) space

def balance\_parens\_1(string):

string = list(string)

stack = []

for index,item in enumerate(string):

if item ==")":

if not stack:

string[index]=""

else:

stack.pop()

elif item =="(":

stack.append(index)

for index in stack:

string[index] = ""

return "".join(string)

#print (balance\_parens\_naive(string))

#print (balance\_parens\_1(string))

#Approach 5: Keep a counter to track parens, 3 passes, mutate original string (O(n) time and O(1) space)

def balance\_parens\_2(string):

countL = 0

string = list(string)

for index,item in enumerate(string):

if item ==")":

if not countL:

string[index]=""

else:

countL-=1

elif item =="(":

countL+=1

countR = 0

# print (string)

for index in range(len(string)-1,-1,-1):

item = string[index]

if item =="(":

if not countR:

# print (index)

string[index]=""

else:

countR-=1

elif item ==")":

countR+=1

return "".join(string)

string = ")(())("

print (balance\_parens\_2(string))class Solution:

def evaluate\_expr(self, stack):

res = stack.pop() if stack else 0

# Evaluate the expression till we get corresponding ')'

while stack and stack[-1] != ')':

sign = stack.pop()

if sign == '+':

res += stack.pop()

else:

res -= stack.pop()

return res

def calculate(self, s: str) -> int:

stack = []

n, operand = 0, 0

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

ch = s[i]

if ch.isdigit():

# Forming the operand - in reverse order.

operand = (10\*\*n \* int(ch)) + operand

n += 1

elif ch != " ":

if n:

# Save the operand on the stack

# As we encounter some non-digit.

stack.append(operand)

n, operand = 0, 0

if ch == '(':

res = self.evaluate\_expr(stack)

stack.pop()

# Append the evaluated result to the stack.

# This result could be of a sub-expression within the parenthesis.

stack.append(res)

# For other non-digits just push onto the stack.

else:

stack.append(ch)

# Push the last operand to stack, if any.

if n:

stack.append(operand)

# Evaluate any left overs in the stack.

return self.evaluate\_expr(stack)# -\*- coding: utf-8 -\*-

"""

Created on Sun Sep 29 11:38:25 2019

@author: Huy Nguyen

"""

#Tree Iterator

#173. Binary Search Tree Iterator

#Implement an iterator over a binary search tree (BST). Your iterator will be initialized with the root node of a BST.

#

#Calling next() will return the next smallest number in the BST.

# Definition for a binary tree node.\

class TreeNode:

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

self.val = x

self.left = left

self.right = right

class ExtendedBSTIterator :

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

self.data= []

self.head = TreeNode(None)

def dfs(root):

if root:

dfs(root.left)

self.data.append(root.val)

dfs(root.right)

dfs(root)

self.index = -1

self.lookBack =0

self.k = k

def next(self) -> int:

"""

@return the next smallest number

"""

if self.hasNext():

self.check = True

self.index+=1

self.lookBack = 0

return self.data[self.index]

def hasNext(self) -> bool:

"""

@return whether we have a next smallest number

"""

return self.index<len(self.data)-1

def hasPrev(self):

return self.index>0 and self.lookBack<self.k

def prev(self):

if self.hasPrev():

self.index-=1

self.lookBack+=1

return self.data[self.index]

#root = TreeNode(7)

#root.left = TreeNode(3)

#root.right = TreeNode(15,TreeNode(9),TreeNode(20))

#iterator = ExtendedBSTIterator(root,3)

#print (iterator.head)

#print (iterator.head.right)

#print (iterator.head.left)

#print (iterator.hasNext())

#print (iterator.next())

#print (iterator.next())

#print (iterator.next())

#print (iterator.next())

#print (iterator.hasPrev())

#print (iterator.prev())

#print (iterator.hasPrev())

#print (iterator.prev())

#print (iterator.hasPrev())

#print (iterator.prev())

#print (iterator.hasPrev())

#print (iterator.prev())

#print (iterator.next())

#print (iterator.next())

#print (iterator.hasNext())

#print (iterator.hasPrev())

#print (iterator.prev())

#print (iterator.hasNext())

#print (iterator.next())

#next() and hasNext() should run in average O(1) time and uses O(h) memory, where h is the height of the tree.

class BSTIterator:

def \_\_init\_\_(self, root: TreeNode):

self.check = True

self.head = TreeNode(None)

self.maxLast = None

def dfs(root):

if root:

dfs(root.left)

if not self.maxLast:

self.head.right = root

else:

self.maxLast.right = root

root.left = self.maxLast

self.maxLast = root

dfs(root.right)

dfs(root)

# print ()

def next(self) -> int:

"""

@return the next smallest number

"""

if self.hasNext():

self.head = self.head.right

return self.head.val

def hasNext(self) -> bool:

"""

@return whether we have a next smallest number

"""

if self.head.right:

return True

return False

root = TreeNode(7)

root.left = TreeNode(3)

root.right = TreeNode(15,TreeNode(9),TreeNode(20))

iterator = BSTIterator(root)

print (iterator.next())

print (iterator.next())

print (iterator.hasNext())

print (iterator.next())

print (iterator.hasNext())

print (iterator.next())

print (iterator.hasNext())

print (iterator.next())

print (iterator.hasNext())

# -\*- coding: utf-8 -\*-

"""

Created on Thu Oct 10 22:51:51 2019

@author: huyn

"""

#Binary Tree Paths

def binaryTreePaths(root):

res = []

def dfs(root,string):

if root:

string.append(str(root.val))

if root.left and root.right:

dfs(root.left,string)

dfs(root.right,string)

elif root.left:

dfs(root.left,string)

elif root.right:

dfs(root.right,string)

else:

res.append("->".join(string))

string.pop()

dfs(root,[])

return res# -\*- coding: utf-8 -\*-

"""

Created on Thu Oct 10 17:13:20 2019

@author: Huy Nguyen

"""

#binary right side view

from collections import deque

def rightSideView(root):

res = []

if not root:

return res

queue = deque([root])

while queue:

# we will append the last node in queue to res

res.append(queue[-1].val)

size = len(queue)

for i in range(size):

node = queue.popleft()

if node.left:

queue.append(node.left)

if node.right:

queue.append(node.right)

return res# -\*- coding: utf-8 -\*-

"""

Created on Sun Sep 29 11:28:17 2019

@author: Huy Nguyen

"""

#Implement Buffer with the following api

#class Buffer {

#

# public Buffer(int capacity) {

# }

#

# /\*\*

# \* Transfers the content of the given source char array into this buffer.

# \* Returns the the number of chars that were written into the buffer.

# \*/

# public int write(char[] src) {

# }

#

# public char[] read(int n) {

# }

#}

from collections import deque

class Buffer:

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

self.capacity = capacity

self.data = deque([])

def write(self,string):

count =0

for i in range(min(len(string),self.capacity-len(self.data))):

self.data.append(string[i])

count+=1

return count

def read(self,n):

string = ""

for i in range(min(n,len(self.data))):

string+=self.data.popleft()

return string

myBuffer = Buffer(5)

print (myBuffer.write("abc"))

print (myBuffer.write("def"))

print (myBuffer.read(3))

print (myBuffer.write("xyzabc"))

print (myBuffer.read(8))

# -\*- coding: utf-8 -\*-

"""

Created on Fri Oct 11 12:39:08 2019

@author: huyn

"""

def calculator():

nums = 0

operations = "+-="

accumulate = None

lastOp =None

while True:

char = input("Please type in your input:")

if char in operations:

if not lastOp:

lastOp = char

# store accumulate as our our num

if accumulate==None:

accumulate = nums

else:

if lastOp=="+":

accumulate+=nums

elif lastOp=="-":

accumulate-=nums

# print out to screen, basically do a lazy evaluation ehre

print (accumulate)

# check if is not equal, then we store as lastOp

if char!="=":

lastOp= char

# we reset our nums after every operation

nums = 0

else: # we store first num into accumulate

nums= nums\*10+int(char)

calculator()

# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 9 03:20:17 2019

@author: huyn

"""

##Cartesian tree

#build a tree with property:

# 1. binary tree

# 2. Min Heap

# 3. In order traversal return array

class TreeNode:

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

self.val = val

self.left = left

self.right = right

def inorder(self,root):

if root:

self.inorder(root.left)

print (root.val)

self.inorder(root.right)

class cartersianTree:

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

self.root = self.initialize(arr)

def initialize(self,arr):

root = None

for item in arr:

newNode = TreeNode(item)

if not root:

root = newNode

else:

if newNode.val>root.val:

# make our root as newNode child, set newNode as root

newNode.left = root

root = newNode

# bigger will be lower, and something happens first will be on the left child

else:

# we traverse the right until we hit nothing, or we hit one that is greater than root

# use a copy of the root

tempRoot = root

while tempRoot.right:

# we have to check if our root.right is still lower than item

if tempRoot.right.val<item:

tempRoot= tempRoot.right

else:

# we have to swap

temp = tempRoot.right

tempRoot.right = newNode

newNode.left = temp

break

# we break the loop either we hit none or we have done assignment, if we have done

# asssignment , our tempRoot.right would not be empty

if not tempRoot.right:

tempRoot.right = newNode

return root

arr = [5,8,6,1]

tree = cartersianTree(arr)

root = tree.root

root.inorder(root)

# O(nlogn),O(n) space# -\*- coding: utf-8 -\*-

"""

Created on Fri Sep 20 11:48:03 2019

@author: huyn

"""

#489. Robot Room Cleaner

#Given a robot cleaner in a room modeled as a grid.

#

#Each cell in the grid can be empty or blocked.

#

#The robot cleaner with 4 given APIs can move forward, turn left or turn right. Each turn it made is 90 degrees.

#

#When it tries to move into a blocked cell, its bumper sensor detects the obstacle and it stays on the current cell.

#

#Design an algorithm to clean the entire room using only the 4 given APIs shown below.

"""

This is the robot's control interface.

You should not implement it, or speculate about its implementation

"""

class Robot:

def \_\_init\_\_(self,matrix,current,index=3):# facing up as initial

self.matrix = matrix

self.current = current

self.directions = [(0,1),(1,0),(0,-1),(-1,0)] # + means we turning right

self.index = index

def move(self):

"""

Returns true if the cell in front is open and robot moves into the cell.

Returns false if the cell in front is blocked and robot stays in the current cell.

:rtype bool

"""

addX,addY = self.directions[self.index]

x,y = self.current

X,Y = x+addX,y+addY

if X>=0 and Y>=0 and X<len(self.matrix) and Y<len(self.matrix[0]):

if self.matrix[X][Y]!=0:

self.current = X,Y

return True

return False

def turnLeft(self):

"""

Robot will stay in the same cell after calling turnLeft/turnRight.

Each turn will be 90 degrees.

:rtype void

"""

if self.index==0:

self.index= 3

else:

self.index-=1

def turnRight(self):

"""

Robot will stay in the same cell after calling turnLeft/turnRight.

Each turn will be 90 degrees.

:rtype void

"""

if self.index==3:

self.index= 0

else:

self.index+=1

def clean(self):

"""

Clean the current cell.

:rtype void

"""

x,y = self.current

self.matrix[x][y]="X" # the clean wont know this

#room = [

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

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

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

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

# [1,1,1,1,1,1,1,1]

#]

#robot =Robot(room,[1,3])

#print (robot.current)

#print (robot.move())

#print (robot.current)

#robot.turnLeft()

#print (robot.move())

#print (robot.current)

#robot.turnRight()

#print (robot.move())

#print (robot.current)

#robot.turnLeft()

#print (robot.move())

#print (robot.current)

#robot.turnLeft()

#print (robot.move())

#print (robot.current)

def cleanRoom(robot:Robot):

"""

:type robot: Robot

:rtype: None

"""

visited= set()

visited.add((0,0))

# we clean the current cell

robot.clean()

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

currentIndex = 3

def proceed(currentCell,currentIndex):

x,y = currentCell

if robot.move():

# compute the cell cordinate

addX,addY = directions[currentIndex]

# check if this cell already fully visited

X,Y = x+addX,y+addY

newCell = (X,Y)

# print ("newCell",newCell)

if (X,Y) not in visited:

# we clean this tile

robot.clean()

dfs(newCell,currentIndex)

# after dfs, we have to go back ward for the robot

# we turn left twice and move

robot.turnLeft()

robot.turnLeft()

robot.move()

robot.turnLeft()

robot.turnLeft()

# now we are back to our currentCell for our robot and with the same direction

# print (127,currentCell)

# this is dfs, basically where we will traverse, keep track,clean the room

def dfs(currentCell,currentIndex):

# we check all 4 cell around currentCell

# we will try to move away from our position 3 times

# move forward

visited.add(currentCell)

x,y = currentCell

# print (133,currentCell,robot.current)

for i in range(4):

proceed(currentCell,currentIndex)

currentIndex= (currentIndex+1)%4

# print (137,currentCell,robot.current)

robot.turnRight()

# if i==1:

# break

dfs((0,0),currentIndex)

print (robot.matrix)

#room =[[1,1,1],[1,1,1],[1,1,1]]

room = [

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

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

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

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

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

]

robot =Robot(room,[2,2])

cleanRoom(robot)

# -\*- coding: utf-8 -\*-

"""

Created on Thu Sep 19 15:08:07 2019

@author: huyn

"""

from collections import deque

class Node:

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

self.val = val

self.neighbors = neighbors

# 133. Clone Graph

#Given a reference of a node in a connected undirected graph, return a deep copy (clone) of the graph.

#Each node in the graph contains a val (int) and a list (List[Node]) of its neighbors.

def cloneGraphDFS(root):

dictionary = {}

def dfs(root):

if root:

# time to create a clone of root

newNode = Node(root.val,[])

dictionary[root] = newNode

# for each neighbor ouf root

for neighbor in root.neighbors:

# create a clone, if neighbor was not traverse

if neighbor not in dictionary:

cloneNeighbor = dfs(neighbor) # return the clone node

# append this to our newNode neighbor

else:

# alredy make a clone ,retrieve it

cloneNeighbor= dictionary[neighbor]

newNode.neighbors.append(cloneNeighbor)

return newNode

return dfs(root)

def cloneGraphLoop(node):

queue = deque()

visited = {}

queue.append(node)

visited[node] = Node(node.val, [])

while queue:

cur = queue.popleft()

for neighbor in cur.neighbors:

if neighbor not in visited:

queue.append(neighbor)

visited[neighbor] = Node(neighbor.val, [])

visited[cur].neighbors.append(visited[neighbor])

return visited[node]# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 2 23:43:24 2019

@author: huyn

"""

#123456789 = K

def target(k):

letter = "123456789"

res = []

def dfs(index,currentVal,k,path):

if index==len(letter):

if currentVal ==k:

res.append("".join(path))

else:

for i in range(index,len(letter)):

val = int(letter[index:i+1])

# -

currentVal-=val

path.append("-")

path.append(letter[index:i+1])

dfs(i+1,currentVal,k,path)

currentVal+=val

path.pop()

path.pop()

# +

currentVal+=val

path.append("+")

path.append(letter[index:i+1])

dfs(i+1,currentVal,k,path)

currentVal-=val

path.pop()

path.pop()

for i in range(len(letter)):

val = int(letter[:i+1])

dfs(i+1,-val,k,["-"+letter[:i+1]])

dfs(i+1,val,k,[letter[:i+1]])

return res

#print (target(100))# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 9 16:51:27 2019

@author: huyn

"""

#3Sum

from typing import List

def threeSum(self, nums: List[int]) -> List[List[int]]:

nums.sort()

res =set()

check = set()

for index,num in enumerate(nums):

if num not in check:

check.add(num)

start,stop = index+1,len(nums)-1

s = -num

while start<stop:

val = nums[start]+nums[stop]

if val==s:

res.add((num,nums[start],nums[stop]))

start+=1

stop-=1

elif val<s:

start+=1

else:

stop-=1

return [list(item) for item in res]# -\*- coding: utf-8 -\*-

"""

Created on Thu Sep 19 11:05:08 2019

@author: huyn

"""

import collections

#721. Accounts Merge

#Given a list accounts, each element accounts[i] is a list of strings, where the first element

#accounts[i][0] is a name, and the rest of the elements are emails representing emails of the account.

#

#Now, we would like to merge these accounts. Two accounts definitely belong to the

#same person if there is some email that is common to both accounts. Note that even if

# two accounts have the same name, they may belong to different people as people could have the same name.

# A person can have any number of accounts initially, but all of their accounts definitely have the same name.

#

#After merging the accounts, return the accounts in the following format: the first

#element of each account is the name, and the rest of the elements are emails in sorted order.

#The accounts themselves can be returned in any order.

def accountsMerge(accounts):

em\_to\_name = {}

graph = collections.defaultdict(set) # create a graph that store infor of emails that are unique

# this graph would help us to do connected components later

for acc in accounts:

name = acc[0]

for email in acc[1:]:

# we dont need an edge between every pair of edge, just add edge between first to every other

# and everyother to first, then later on, if appear an edge to another different email

# we can still join them together

graph[acc[1]].add(email)

graph[email].add(acc[1])

# link the email back to the name

em\_to\_name[email] = name

seen = set()

ans = []

for email in graph:

# we do a connected component search here

if email not in seen:

# we add this email to seen

seen.add(email)

# initiate a stack for our component

stack = [email]

component = []

while stack:

node = stack.pop()

component.append(node)

for nei in graph[node]:

if nei not in seen:

seen.add(nei)

stack.append(nei)

# done w stack means we have connected all possible that link to the email, using the map to name

# to create an entry in answer

ans.append([em\_to\_name[email]] + sorted(component))

return ans

#accounts = [["John", "johnsmith@mail.com", "john00@mail.com"], ["John", "johnnybravo@mail.com"], ["John", "johnsmith@mail.com", "john\_newyork@mail.com"], ["Mary", "mary@mail.com"]]

#print (accountsMerge(accounts))

# -\*- coding: utf-8 -\*-

"""

Created on Thu Oct 3 00:44:38 2019

@author: huyn

"""

#67. Add Binary

def addBinary(self, a: str, b: str) -> str:

remainder = 0

res = []

i,j =len(a)-1,len(b)-1

while i>=0 and j>=0:

valA = int(a[i])

valB = int(b[j])

if valA+valB+remainder==3:

res.append("1")

remainder = 1

elif valA+valB+remainder ==2:

res.append("0")

remainder = 1

else:

res.append(str(valA+valB+remainder))

remainder = 0

i-=1

j-=1

while i>=0:

valA = int(a[i])

if valA+remainder==2:

res.append("0")

remainder = 1

else:

res.append(str(valA+remainder))

remainder = 0

i-=1

while j>=0:

valB = int(b[j])

if valB+remainder==2:

res.append("0")

remainder = 1

else:

res.append(str(valB+remainder))

remainder = 0

j-=1

if remainder:

res.append("1")

return "".join(res[::-1])# -\*- coding: utf-8 -\*-

"""

Created on Sun Sep 22 18:13:39 2019

@author: huyn

"""

import random

#282. Expression Add Operators

#Given a string that contains only digits 0-9 and a target value, return all

#possibilities to add binary operators (not unary) +, -, or \* between the digits so they evaluate to the target value.

def addOperators(num: str, target: int):

arr = []

def dfs(start,path,target,accumulate,prevVal):

# print (path,accumulate)

if start == len(num):

if accumulate==target:

arr.append("".join(path))

elif start<len(num):

# print (path)

for i in range(start,len(num)):

string = num[start:i+1]

# check if string is valid

if str(int(string))==string:

try:

val = int(string)

except:

val = 0

path.append("+")

path.append(string)

dfs(i+1,path,target,accumulate+val,val)

path.pop()

path.pop()

path.append("-")

path.append(string)

dfs(i+1,path,target,accumulate-val,-val)

path.pop()

path.pop()

path.append("\*")

path.append(string)

pre = prevVal\*val

dfs(i+1,path,target,accumulate-prevVal+pre,pre)

path.pop()

path.pop() # pop trhe string

for i in range(len(num)):

string = num[:i+1]

if str(int(string))==string:

dfs(i+1,[string],target,int(string),int(string))

return arr

#num ="12345"

#print (addOperators(num,4))

#target = 6

#num = "123"

#print (addOperators(num,target))

#target = 6

#num = "232"

#print (addOperators(num,target))

#target = 8

#num = "232"

#print (addOperators(num,target))

#target = 0

#num = "00"

#print (addOperators(num,target))

for i in range(1):

num = random.randint(1,10\*\*6)

target = random.randint(1,10)

print (""""{}"\n{}""".format(num,target))# -\*- coding: utf-8 -\*-

"""

Created on Thu Oct 10 00:16:14 2019

@author: Huy Nguyen

"""

def addTwoNumbers(self, l1: ListNode, l2: ListNode) -> ListNode:

def dfs(l1,l2,over):

if l1 and l2:

n1 = l1.val

n2 = l2.val

val =n1+n2+over

newNode = ListNode(val%10)

newNode.next = dfs(l1.next,l2.next,val//10)

return newNode

elif l1:

n1 = l1.val

val =n1+over

newNode = ListNode(val%10)

newNode.next = dfs(l1.next,l2,val//10)

return newNode

elif l2:

n2 = l2.val

val =n2+over

newNode = ListNode(val%10)

newNode.next = dfs(l1,l2.next,val//10)

return newNode

else:

if over==1:

return ListNode(1)

return dfs(l1,l2,0)# -\*- coding: utf-8 -\*-

"""

Created on Sun Sep 22 18:43:09 2019

@author: huyn

"""

#269. Alien Dictionary

#There is a new alien language which uses the latin alphabet. However, the order

# among letters are unknown to you. You receive a list of non-empty words from the dictionary,

# where words are sorted lexicographically by the rules of this new language.

# Derive the order of letters in this language.

def alienOrder(words):

res = []

parents = {} # map the backward edge to check whether a character should be a start node

neighbors = {} # store the forward edge

for word in words:

for letter in word:

if letter not in parents:

parents[letter]=set()

if letter not in neighbors:

neighbors[letter] = set()

#

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

w1 = words[i]

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

w2 = words[j]

for k in range(min(len(w1),len(w2))):

if w1[k]==w2[k]:

continue

elif w1[k]!=w2[k]:

smaller = w1[k]

larger = w2[k]

# set up neighbor that map smaller to larger

# make parents as set so we can remove the parent from each node faster

neighbors[smaller].add(larger)

parents[larger].add(smaller)

# remember to break

break

startNodes = [node for node in parents if len(parents[node])==0]

# we keep pop out startnodes

while startNodes:

node = startNodes.pop()

# add this to res, we know for sure it is equal or less to everything on the left

res.append(node)

# for each neighbor, remove edge from neighbor to node as parent

for neighbor in neighbors[node]:

# remove edge

parents[neighbor].remove(node)

# if there are no more parents of neighbor, add it to start node

if not parents[neighbor]:

startNodes.append(neighbor)

# we check if there are some edges for any key in parents node

for node in parents:

if parents[node]:

return ""

return "".join(res)

#words = ["ab","adc"]

#words = ["z","z"]

#print (alienOrder(words))

# -\*- coding: utf-8 -\*-

"""

Created on Tue Oct 8 15:34:55 2019

@author: huyn

"""

#Amazing Number

#Define amazing number as: its value is less than or equal to its index. Given a

#circular array, find the starting position, such that the total number of amazing numbers in the array is maximized.

#Example 1: 0, 1, 2, 3

#Ouptut: 0. When starting point at position 0, all the elements in the array are equal to

#its index. So all the numbers are amazing number.

#Example 2: 1, 0 , 0

#Output: 1. When starting point at position 1, the array becomes 0, 0, 1. All the elements are amazing number.

#If there are multiple positions, return the smallest one.

#

#should get a solution with time complexity less than O(N^2)

def getAmazingNumberNaive(arr):

size = len(arr)

maxCount = 0

for i in range(size):

starting =i

count = 0

for j in range(size):

# print (starting,j)

if j>=arr[starting%size]:

# print ("index:",j, "num:",arr[starting%size])

count+=1

starting+=1

maxCount= max(maxCount,count)

return maxCount

arr =[0,1,2,3]

print (getAmazingNumberNaive(arr))

#arr=[3,2,1,0]

#

#print (getAmazingNumberNaive(arr))

#arr = [4]

#print (getAmazingNumberNaive(arr))# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 9 16:23:36 2019

@author: huyn

"""

#myAtoi

def myAtoi(string: str) -> int:

numeric = []

d ={'7': 7, '3': 3, '8': 8, '6': 6, '5': 5, '2': 2, '1': 1, '4': 4, '9': 9, '0': 0}

for letter in string:

if letter.isalpha():

break

elif letter ==" " and not numeric:

continue

elif letter ==" " and numeric:

break

elif letter.isnumeric():

numeric.append(letter)

elif (letter =="-" or letter =="+") and not numeric:

if letter =="-" :

numeric.append("-")

else:

numeric.append("+")

else:

break

isNeg= False

if not numeric:

return 0

if numeric[0]=="-" or numeric[0]=="+":

if numeric[0]=="-":

isNeg= True

numeric.pop(0)

# we go to the first numeric that is not 0

while numeric:

if numeric[0]=="0":

numeric.pop(0)

else:

break

val = 0

while numeric:

val=val\*10+d[numeric.pop(0)]

if isNeg:

return max(-2147483648,-val)

return min(val,2147483647)# -\*- coding: utf-8 -\*-

"""

Created on Thu Sep 26 14:44:18 2019

@author: huyn

"""

#Balance the parenthesis in a string

#You are given a string with alphanumeric characters and parentheses. Your goal

#is to return a string with balanced parentheses by removing the fewest characters possible.

#Note that you cannot add anything to the string.

#"()" -> "()"

#"b(a)r)" -> "b(a)r"

#")(" -> ""

#"(((((" -> ""

#")(())(" -> "(())"

#string = ")(())("

def balance\_parens\_naive(string):

stack = []

indices = []

for index,item in enumerate(string):

if item ==")":

if not stack:

indices.append(index)

else:

stack.pop()

elif item =="(":

stack.append(index)

indices.extend(stack)

return "".join([string[i] for i in range(len(string)) if i not in indices])

# O(n) time, O(n) space

def balance\_parens\_1(string):

string = list(string)

stack = []

for index,item in enumerate(string):

if item ==")":

if not stack:

string[index]=""

else:

stack.pop()

elif item =="(":

stack.append(index)

for index in stack:

string[index] = ""

return "".join(string)

#print (balance\_parens\_naive(string))

#print (balance\_parens\_1(string))

#Approach 5: Keep a counter to track parens, 3 passes, mutate original string (O(n) time and O(1) space)

def balance\_parens\_2(string):

countL = 0

string = list(string)

for index,item in enumerate(string):

if item ==")":

if not countL:

string[index]=""

else:

countL-=1

elif item =="(":

countL+=1

countR = 0

# print (string)

for index in range(len(string)-1,-1,-1):

item = string[index]

if item =="(":

if not countR:

# print (index)

string[index]=""

else:

countR-=1

elif item ==")":

countR+=1

return "".join(string)

string = ")(())("

print (balance\_parens\_2(string))class Solution:

def evaluate\_expr(self, stack):

res = stack.pop() if stack else 0

# Evaluate the expression till we get corresponding ')'

while stack and stack[-1] != ')':

sign = stack.pop()

if sign == '+':

res += stack.pop()

else:

res -= stack.pop()

return res

def calculate(self, s: str) -> int:

stack = []

n, operand = 0, 0

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

ch = s[i]

if ch.isdigit():

# Forming the operand - in reverse order.

operand = (10\*\*n \* int(ch)) + operand

n += 1

elif ch != " ":

if n:

# Save the operand on the stack

# As we encounter some non-digit.

stack.append(operand)

n, operand = 0, 0

if ch == '(':

res = self.evaluate\_expr(stack)

stack.pop()

# Append the evaluated result to the stack.

# This result could be of a sub-expression within the parenthesis.

stack.append(res)

# For other non-digits just push onto the stack.

else:

stack.append(ch)

# Push the last operand to stack, if any.

if n:

stack.append(operand)

# Evaluate any left overs in the stack.

return self.evaluate\_expr(stack)# -\*- coding: utf-8 -\*-

"""

Created on Sun Sep 29 11:38:25 2019

@author: Huy Nguyen

"""

#Tree Iterator

#173. Binary Search Tree Iterator

#Implement an iterator over a binary search tree (BST). Your iterator will be initialized with the root node of a BST.

#

#Calling next() will return the next smallest number in the BST.

# Definition for a binary tree node.\

class TreeNode:

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

self.val = x

self.left = left

self.right = right

class ExtendedBSTIterator :

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

self.data= []

self.head = TreeNode(None)

def dfs(root):

if root:

dfs(root.left)

self.data.append(root.val)

dfs(root.right)

dfs(root)

self.index = -1

self.lookBack =0

self.k = k

def next(self) -> int:

"""

@return the next smallest number

"""

if self.hasNext():

self.check = True

self.index+=1

self.lookBack = 0

return self.data[self.index]

def hasNext(self) -> bool:

"""

@return whether we have a next smallest number

"""

return self.index<len(self.data)-1

def hasPrev(self):

return self.index>0 and self.lookBack<self.k

def prev(self):

if self.hasPrev():

self.index-=1

self.lookBack+=1

return self.data[self.index]

#root = TreeNode(7)

#root.left = TreeNode(3)

#root.right = TreeNode(15,TreeNode(9),TreeNode(20))

#iterator = ExtendedBSTIterator(root,3)

#print (iterator.head)

#print (iterator.head.right)

#print (iterator.head.left)

#print (iterator.hasNext())

#print (iterator.next())

#print (iterator.next())

#print (iterator.next())

#print (iterator.next())

#print (iterator.hasPrev())

#print (iterator.prev())

#print (iterator.hasPrev())

#print (iterator.prev())

#print (iterator.hasPrev())

#print (iterator.prev())

#print (iterator.hasPrev())

#print (iterator.prev())

#print (iterator.next())

#print (iterator.next())

#print (iterator.hasNext())

#print (iterator.hasPrev())

#print (iterator.prev())

#print (iterator.hasNext())

#print (iterator.next())

#next() and hasNext() should run in average O(1) time and uses O(h) memory, where h is the height of the tree.

class BSTIterator:

def \_\_init\_\_(self, root: TreeNode):

self.check = True

self.head = TreeNode(None)

self.maxLast = None

def dfs(root):

if root:

dfs(root.left)

if not self.maxLast:

self.head.right = root

else:

self.maxLast.right = root

root.left = self.maxLast

self.maxLast = root

dfs(root.right)

dfs(root)

# print ()

def next(self) -> int:

"""

@return the next smallest number

"""

if self.hasNext():

self.head = self.head.right

return self.head.val

def hasNext(self) -> bool:

"""

@return whether we have a next smallest number

"""

if self.head.right:

return True

return False

root = TreeNode(7)

root.left = TreeNode(3)

root.right = TreeNode(15,TreeNode(9),TreeNode(20))

iterator = BSTIterator(root)

print (iterator.next())

print (iterator.next())

print (iterator.hasNext())

print (iterator.next())

print (iterator.hasNext())

print (iterator.next())

print (iterator.hasNext())

print (iterator.next())

print (iterator.hasNext())

# -\*- coding: utf-8 -\*-

"""

Created on Thu Oct 10 22:51:51 2019

@author: huyn

"""

#Binary Tree Paths

def binaryTreePaths(root):

res = []

def dfs(root,string):

if root:

string.append(str(root.val))

if root.left and root.right:

dfs(root.left,string)

dfs(root.right,string)

elif root.left:

dfs(root.left,string)

elif root.right:

dfs(root.right,string)

else:

res.append("->".join(string))

string.pop()

dfs(root,[])

return res# -\*- coding: utf-8 -\*-

"""

Created on Thu Oct 10 17:13:20 2019

@author: Huy Nguyen

"""

#binary right side view

from collections import deque

def rightSideView(root):

res = []

if not root:

return res

queue = deque([root])

while queue:

# we will append the last node in queue to res

res.append(queue[-1].val)

size = len(queue)

for i in range(size):

node = queue.popleft()

if node.left:

queue.append(node.left)

if node.right:

queue.append(node.right)

return res# -\*- coding: utf-8 -\*-

"""

Created on Sun Sep 29 11:28:17 2019

@author: Huy Nguyen

"""

#Implement Buffer with the following api

#class Buffer {

#

# public Buffer(int capacity) {

# }

#

# /\*\*

# \* Transfers the content of the given source char array into this buffer.

# \* Returns the the number of chars that were written into the buffer.

# \*/

# public int write(char[] src) {

# }

#

# public char[] read(int n) {

# }

#}

from collections import deque

class Buffer:

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

self.capacity = capacity

self.data = deque([])

def write(self,string):

count =0

for i in range(min(len(string),self.capacity-len(self.data))):

self.data.append(string[i])

count+=1

return count

def read(self,n):

string = ""

for i in range(min(n,len(self.data))):

string+=self.data.popleft()

return string

myBuffer = Buffer(5)

print (myBuffer.write("abc"))

print (myBuffer.write("def"))

print (myBuffer.read(3))

print (myBuffer.write("xyzabc"))

print (myBuffer.read(8))

# -\*- coding: utf-8 -\*-

"""

Created on Fri Oct 11 12:39:08 2019

@author: huyn

"""

def calculator():

nums = 0

operations = "+-="

accumulate = None

lastOp =None

while True:

char = input("Please type in your input:")

if char in operations:

if not lastOp:

lastOp = char

# store accumulate as our our num

if accumulate==None:

accumulate = nums

else:

if lastOp=="+":

accumulate+=nums

elif lastOp=="-":

accumulate-=nums

# print out to screen, basically do a lazy evaluation ehre

print (accumulate)

# check if is not equal, then we store as lastOp

if char!="=":

lastOp= char

# we reset our nums after every operation

nums = 0

else: # we store first num into accumulate

nums= nums\*10+int(char)

calculator()

# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 9 03:20:17 2019

@author: huyn

"""

##Cartesian tree

#build a tree with property:

# 1. binary tree

# 2. Min Heap

# 3. In order traversal return array

class TreeNode:

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

self.val = val

self.left = left

self.right = right

def inorder(self,root):

if root:

self.inorder(root.left)

print (root.val)

self.inorder(root.right)

# -\*- coding: utf-8 -\*-

"""

Created on Thu Oct 10 15:24:34 2019

@author: Huy Nguyen

"""

class Node:

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

self.val = val

self.next = next

self.random = random

#Copy List with Random Pointer

def copyRandomList(head: 'Node') -> 'Node':

dictionary = {}

def dfs(head):

if head:

if head not in dictionary: #if we had not map this guy

newLinkNode = Node(head.val,None,None)

dictionary[head] = newLinkNode

# we try to link our newLinkNode to the node that come next from head

nextNode = dfs(head.next)

newLinkNode.next = nextNode# linking

# we also try to link our newLinkNode to the random node lol

nextRandomNode = dfs(head.random)

newLinkNode.random = nextRandomNode

return newLinkNode

else:

# if head already in, then we just return the copy of this head

return dictionary[head]

else:

return None

return dfs(head)

# -\*- coding: utf-8 -\*-

"""

Created on Mon Oct 7 12:30:11 2019

@author: huyn

"""

class TreeNode:

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

self.val= val

self.left = left

self.right = right

#222. Count Complete Tree Nodes

#Given a complete binary tree, count the number of nodes.

def countNodes(root: TreeNode) -> int:

if not root:

return 0

left = countNodes(root.left)

right = countNodes(root.right)

return 1+left+right

a = TreeNode(3)

c = TreeNode(5,a)

d = TreeNode(6)

e = TreeNode(7)

f = TreeNode(8,d,e)

g = TreeNode(10,c,f)

# since we know it is a full binary tree, traverse to the left until we can't traverse more, then try to count node at that step

def countNodesSmart(root):

def getDepth(root):

if not root:

return 0

return 1+getDepth(root.left)

depth = getDepth(root)

def findPath(index):

path = []

while index:

path.append(index%2)

index//=2

return path

def checkIndex(root,path):

for index in range(len(path)-1,-1,-1):

if index==1:

if root.right:

root= root.right

else:

return False

else:

if root.left:

root = root.left

else:

return False

return True

if depth<=1:

return depth

start = 0

stop = 2\*\*depth-1

while start+1<stop:

mid = (start+stop)//2

path = findPath(mid)

if checkIndex(root,path):

start = mid

else:

stop = mid

if checkIndex(root,findPath(stop)):

return 2\*\*(depth-1)+stop

elif checkIndex(root,findPath(start)):

return 2\*\*(depth-1)+start

print (countNodesSmart(g))

# -\*- coding: utf-8 -\*-

"""

Created on Mon Sep 30 11:07:54 2019

@author: huyn

"""

from collections import deque

#Palindromic Subsequences

#Given a string s, return all palindromic subsequences of s.

def findPalindromicSubsequencesNaive(s):

res=deque([""])

for letter in s:

size = len(res)

for i in range(size):

item = res.popleft()

res.append(item+letter)

res.append(item)

return [item for item in res if checkPalindrome(item)]

def checkPalindrome(string):

if not string:

return False

for i in range(len(string)//2):

if string[i]!=string[len(string)-1-i]:

return False

return True

#s = "abac"

#print (findPalindromicSubsequencesNaive(s))

def findPalindromeDP(s):

res = []

return res

s='abcdabcdabcdabcdabcdabcdabcdabcddcbadcbadcbadcbadcbadcbadcbadcba'# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 9 03:21:44 2019

@author: huyn

"""

#Count subsets

#Input:

#

#Given an array A of

#-positive

#-sorted

#-no duplicate

#-integer

#

#A positive integer k

#

#Output:

#

#Count of all such subsets of A,

#Such that for any such subset S,

#Min(S) + Max(S) = k

#subset should contain atleast two elements

def countSubsets(array,k):

d = {}

# we map index to index

count = 0

for index,num in enumerate(array):

d[num]= index

for index1,num in enumerate(array):

val = k-num

index2 = d[val]# since they are unique, index2 and index1 only equal if 2\*num == k

if index1>index2:

break

if index1!=index2:

size =index2-index1-1

count+=2\*\*(size)

return count

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

k = 5

#print (countSubsets(array,k))# -\*- coding: utf-8 -\*-

"""

Created on Sun Sep 29 23:11:29 2019

@author: Huy Nguyen

"""

#Cut Wood

#Given an int array wood representing the length of n pieces of wood and an int k.

# It is required to cut these pieces of wood such that more or equal to k pieces of the same

# length len are cut. What is the longest len you can get?

def cutWood(woods,k):

start = 1

stop = max(woods)

while start+1<stop:

mid = (start+stop)//2

val = check(woods,mid)

if val>=k:

start = mid

else:

stop = mid

if check(woods,stop)>=k:

return stop

if check(woods,start)>=k:

return start

return 0

def check(woods,num):

c =0

for wood in woods:

c+=wood//num

return c

#woods =[ 5, 9, 7]

#k= 3

#print (cutWood(woods,k))

#wood = [5, 9, 7]

#k = 4

#print (cutWood(woods,k))

woods= [1, 2, 3]

k=7

print (cutWood(woods,k))

#woods= [232, 124, 456]

#k = 7

#print (cutWood(woods,k))# -\*- coding: utf-8 -\*-

"""

Created on Tue Sep 17 15:43:00 2019

@author: huyn

"""

#91. Decode Ways

#A message containing letters from A-Z is being encoded to numbers using the following mapping:

#'A' -> 1

#'B' -> 2

#...

#'Z' -> 26

#Given a non-empty string containing only digits, determine the total number of ways to decode it

def numDecodings(s: str) -> int:

if not s:

return 0

if s[0]=="0":

return 0

if len(s)==1:

return 1

arr= [0,0]

if check(s[-1]):

arr[0]+=1

if check(s[-2]) and arr[0]:

arr[1]+=1

if check(s[-2]+s[-1]):

arr[1]+=1

print (arr)

for i in range(len(s)-3,-1,-1):

val =0

if check(s[i]+s[i+1]) and arr[0]:

val+=arr[0]

if check(s[i]) and arr[1]:

val+=arr[1]

arr[0],arr[1]=arr[1],val

return arr[1]

# check function to verify a valid step

def check(string):

# print (string)

if int(string)>26 or int(string)==0:

return False

if "0" in string:

return string in ["10","20"]

return True

#string = "12312731"

#print (numDecodings(string))

#string = "1232312321212"

#print (numDecodings(string))# -\*- coding: utf-8 -\*-

"""

Created on Mon Oct 7 15:46:17 2019

@author: huyn

"""

#498. Diagonal Traverse

#Given a matrix of M x N elements (M rows, N columns), return all elements of the

#matrix in diagonal order as shown in the below image.

from typing import List

def findDiagonalOrder(matrix: List[List[int]]) -> List[int]:

row = len(matrix)

if row ==0 or matrix== [[]]:

return []

col = len(matrix[0])

res = [matrix[0][0]]

currentRow,currentCol = 0,0

rightDown = [(0,1),(1,0)]

diagonal = [(1,-1),(-1,1)]

isRightDown = True

indexR = 0

indexD = 0

count = 0

while count<row\*col-1:

if isRightDown:

# we try both at indexR and indexR+1 to see which one works first

x,y = rightDown[indexR]

if isValid(row,col,currentRow+x,currentCol+y):

currentRow,currentCol= currentRow+x,currentCol+y

res.append(matrix[currentRow][currentCol])

count+=1

else:

x,y = rightDown[(indexR+1)%2]

currentRow,currentCol= currentRow+x,currentCol+y

res.append(matrix[currentRow][currentCol])

count+=1

indexR = (indexR+1)%2

isRightDown = False

else:

x,y = diagonal[indexD]

if isValid(row,col,currentRow+x,currentCol+y):

currentRow,currentCol= currentRow+x,currentCol+y

res.append(matrix[currentRow][currentCol])

count+=1

else: # this means we have change isRIghtDown to True

isRightDown= True

# we also have to increment indexD

indexD = (indexD+1)%2

return res

def isValid(row,col,x,y):

return x>=0 and y>=0 and x<row and y<col

matrix=[

[ 1, 2 ],

[ 4, 5 ],

[ 7, 8 ],

[10,11]

]

#print (findDiagonalOrder(matrix))

#Given a matrix, return all elements of the matrix in antidiagonal order as shown in the below image.

def antiDiagonalOrder(matrix):

row = len(matrix)

col = len(matrix[0])

res = []

coordinate = []

for c in range(col):

res.append([matrix[0][c]])

coordinate.append([0,c])

for r in range(1,row):

res.append([matrix[r][col-1]])

coordinate.append([r,col-1])

# print (res,coordinate)

for i in range(len(coordinate)):

currentRow,currentCol = coordinate[i]

count = 0

while True:

nextRow,nextCol = currentRow+1,currentCol-1

if isValid(row,col,nextRow,nextCol):

currentRow,currentCol= nextRow,nextCol

res[i].append(matrix[currentRow][currentCol])

count+=1

if count ==10:

break

else:

break

# print (item)

return res

#print (antiDiagonalOrder(matrix))# -\*- coding: utf-8 -\*-

"""

Created on Fri Oct 11 00:31:12 2019

@author: huyn

"""

class TreeNode:

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

self.val = x

self.left = None

self.right = None

#Diameter of Binary Tree

def diameterOfBinaryTree(root: TreeNode) -> int:

myMax = [0]

def dfs(root,myMax):

if not root:

return 0

else:

# at root, we look at the path from left, and path from right to see which one has the longer path

left = dfs(root.left,myMax)

right = dfs(root.right,myMax)

currentLength = 1+left+right

myMax[0] = max(myMax[0],currentLength)

return 1+max(left,right)

dfs(root,myMax)

return max(myMax[0]-1,0)

a = TreeNode(1)

b = TreeNode(1)

c = TreeNode(1)

d = TreeNode(1)

e = TreeNode(1)

f = TreeNode(1)

g = TreeNode(1)

a.left = b

a.right = c

b.left = d

b.right = e

c.right = f

f.right = g

#print (diameterOfBinaryTree(a))# -\*- coding: utf-8 -\*-

"""

Created on Wed Sep 25 15:05:48 2019

@author: huyn

"""

#There are two parts to this question. The first is a function called setup that

#takes a list of words. In this function, you have a chance to preprocess the list of words

#in order to solve and improve the runtime of the second part which is a function called isMember.

#isMember takes in a string and returns whether or not that string exists in the list of words.

#isMember may also contain one or more dots (.) which is a wildcard that matches exactly one character

#of any value in setup at the current index of the string.

#Examples

#

#setup(["foo", "bar", "baz"]);

#isMember("foo"); # returns true

#isMember("garply"); # returns false because "garply" is not in the dictionary

#isMember("f.o"); # returns true (it matches foo where the '.' matches the first 'o')

#isMember(".."); # returns false (there are no two-letter words)

#Clarifying questions

#

#setup is a one time call and isMember is called many times. So while time spent

#is setup should be reasonable, time spent in isMember is the most important thing.

#Only lowercase alphabetical characters will exist in the list of words in setup.

# The argument to isMember is also only lowercase alphabetical characters as well as the dot

# (.) representing the wildcard.

# support keep add e the dictionary, keep count howmany with this index

class Trie:

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

self.root = {}

def addWord(self,word):

root = self.root

for letter in word:

if letter not in root:

root[letter] = {}

if "?" not in root:

root["?"]=0

root["?"]+=1

root = root[letter]

# not chunking if there is only

root["#"]=1

root["?"]=1

# we intialize count down here

def search(self,word):

def dfs(word,index,currentRoot):

if index==len(word):

if "#" in currentRoot:

return True

else:

return False

elif index<len(word):

letter = word[index]

if letter!=".":

if letter in currentRoot:

return dfs(word,index+1,currentRoot[letter])

else:

return False

else:

for possibleLetter in currentRoot:

if letter !="#":

if dfs(word,index+1,currentRoot[possibleLetter]):

return True

return False

return dfs(word,0,self.root)

def count(self,suffix):

def dfs(word,index,currentRoot):

if index == len(word):

return currentRoot["?"]

elif index<len(word):

count = 0

letter = word[index]

if letter!=".":

if letter in currentRoot:

count+=dfs(word,index+1,currentRoot[letter])

else:

# print ("currentRoot",currentRoot)

for possibleLetter in currentRoot:

if possibleLetter not in "?#":

# pawn a thread

# print ("possibleLetter",possibleLetter)

count+= dfs(word,index+1,currentRoot[possibleLetter])

return count

return dfs(suffix,0,self.root)

# function that takes in a list of string, return a Trie object that store info

def setup(wordList):

myTrie = Trie()

for word in wordList:

myTrie.addWord(word)

return myTrie

myTrie = setup(["foo", "bar", "baz","abcdfegre","footootoolo","fool","foutain","faol"])

print (myTrie.count("b.."))

print (myTrie.count("f.."))

print (myTrie.count("fo."))

print (myTrie.count("f..l"))

# function search that takes in a word and check if it is store in my Trie

def isMember(word,myTrie):

# a function that dfs through myTrie and find a path that might return True

def dfs(word,index,currentRoot):

if index==len(word):

if "#" in currentRoot:

return True

else:

return False

elif index<len(word):

letter = word[index]

if letter!=".":

if letter in currentRoot:

return dfs(word,index+1,currentRoot[letter])

else:

return False

else:

for possibleLetter in currentRoot:

if letter !="#":

if dfs(word,index+1,currentRoot[possibleLetter]):

return True

return False

return dfs(word,0,myTrie.root)

#print (isMember("foo",myTrie))

#print (isMember("garply",myTrie))

#print (isMember("f.o",myTrie))

#print (isMember("..",myTrie))

#print (isMember("",myTrie))

# class trie that will chunk word together, might not support update

class TrieChunking:

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

self.root = {}

def addWord(self,word):

def dfs(dictionary,word,index):

return

# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 9 14:31:03 2019

@author: huyn

"""

#29. Divide Two Integers

#Given two integers dividend and divisor, divide two integers without using multiplication,

#division and mod operator.

#

#Return the quotient after dividing dividend by divisor.

#

#The integer division should truncate toward zero.

def divideNaive(dividend,divisor):

c = 0

if dividend==0:

return 0

if divisor == 1:

return dividend

if divisor ==-1:

return -dividend

a,b = abs(dividend),abs(divisor)

while a>=b:

c+=1

a-=b

if dividend>0 and divisor>0 or dividend<0 and divisor<0:

return c

else:

return -c

#print (divideNaive(5,3))

#print (divideNaive(10,3))

#print (divideNaive(5,2))

#print (divideNaive(-12,3))

#print (divideNaive(-12,1))

def divideBinary(dividend,divisor):

if dividend<0:

return -divideBinary(-dividend,divisor)

if divisor <0:

return -divideBinary(dividend,-divisor)

if divisor>dividend:

return 0

if dividend ==0:

return 0

if divisor ==1:

return dividend

# -\*- coding: utf-8 -\*-

"""

Created on Mon Oct 7 11:40:39 2019

@author: huyn

"""

#Given the CSV with the belowing information :

#StartTime,user1,user2,action

#There are 4 possible actions : REQUEST,ACCEPT,REJECT,REMOVE

#REMOVE is only possible when users already be friend.

#

#Going through the csv, determine the list of users who are friends.

import csv

def findFriends(csvFile):

data,users = readCSV(csvFile)

graph = makeGraph(data,users)

res = []

for user1 in users:

temp =[user1]

for user2 in users:

if graph[user1][user2] == "F":

temp.append(user2)

res.append(temp)

return res

def readCSV(csvFile):

res = []

users = set()

with open(csvFile) as csv\_file:

csv\_reader = csv.reader(csv\_file)

for row in csv\_reader:

res.append(row)

users.add(row[0])

users.add(row[1])

return sorted(res,key = lambda x:x[0]),users

def makeGraph(data,users):

d ={}

for user1 in users:

for user2 in users:

d[user1][user2]= ""

for time,user1,user2,action in data:

if action == "REQUEST":

d[user1][user2]="R"

elif action=="ACCEPT":

# only make friend if user1 requested user2

if d[user2][user1]=="R":

d[user1][user2]="F"

d[user2][user1]="F"

elif action=="REJECT":

if d[user2][user1]=="R":

d[user1][user2]=""

d[user2][user1]=""

elif action == "REMOVE":

if d[user1][user2]=="F":

d[user1][user2]==""

d[user2][user1]==""

return d

# -\*- coding: utf-8 -\*-

"""

Created on Thu Sep 26 00:47:49 2019

@author: huyn

"""

import math

#Find intersection of 2 sorted interger arrays

def findIntersectionPointers(arr1,arr2):

i,j =0,0

arr = [ ]

while i<len(arr1) and j<len(arr2):

if arr1[i]==arr[j]:

arr.append(arr1[i])

i+=1

j+=1

if arr1[i]<arr2[j]:

i+=1

else:

j+=1

return arr

def findIntersectionBST(arr1,arr2):

if len(arr1)<=len(arr2):

short = arr1

long = arr2

else:

short = arr2

long = arr1

arr = []

start = 0

stop = len(long)-1

for number in short:

index = searchIndex(start,stop,long,number)

# print (number,index,start,stop)

if index == stop and long[index]!=number:

break

if long[index]==number:

arr.append(number)

start = index

if start==len(long):

break

return arr

def searchIndex(start,stop,long,number):

while start+1<stop:

mid = (start+stop)//2

if long[mid]>=number:

stop = mid

else:

start = mid

return stop

#arr1,arr2=[1,3,6,17,21,21,21], [2,4,6,21,21,23]

#arr = findIntersectionBST(arr1,arr2)

# we keep checking the length of the 2 arr

def findIntersectionMix(arr1,arr2):

arr = []

i,j,c1,c2= 0,0,0,0

while i<len(arr1) and j<len(arr2):

if arr1[i]<arr2[j]:

i+=1

c1+=1

elif arr1[i]>arr2[j]:

j+=1

c2+=1

else:

arr.append(arr1[i])

i+=1

j+=1

# check if arr1 is denser than arr2

if c1>math.log(len(arr1),2):

i= searchIndex(i,len(arr1),arr1,arr2[j])

c1= 0

elif c2>math.log(len(arr2),2):

j = searchIndex(j,len(arr2),arr2,arr1[i])

c2 = 0

return arr

arr1,arr2=[1,3,6,17,21,21,21], [2,4,6,21,21,23]

print (findIntersectionMix(arr1,arr2))# -\*- coding: utf-8 -\*-

"""

Created on Mon Oct 7 12:59:10 2019

@author: huyn

"""

#Leftmost column index of 1

#In a binary matrix (all elements are 0 and 1), every row is sorted in ascending

#order (0 to the left of 1). Find the leftmost column index with a 1 in it.

arr = [[0, 0, 0, 1],

[0, 1, 1, 1],

[0, 0, 0, 0],

[0, 0, 0, 0]]

def findLeftMostBST(arr):

row = len(arr)

col = len(arr[0])

stop = col-1

myMin = None

for r in range(row):

if myMin==None:

stop = leftMost(arr[r],col-1)

# print (stop)

else:

stop = leftMost(arr[r],myMin)

# print (stop)

if stop!=-1:

myMin = stop

return myMin

def leftMost(nums,stop):

start =0

# print (nums)

while start+1<stop:

mid = (start+stop)//2

if nums[mid]==1:

stop = mid

# print (stop)

else:

start = mid

if nums[start]==1:

return start

if nums[stop]==1:

return stop

return -1

#print (findLeftMostBST(arr))

def findLeftMostPointers(arr):

row = len(arr)

col = len(arr[0])

r = 0

c = col-1

myMin = -1

while r<row and c>=0:

val =arr[r][c]

if val==0: # we can ignore this row

r+=1

else:

myMin = c

c-=1

return myMin

print (findLeftMostPointers(arr))

# -\*- coding: utf-8 -\*-

"""

Created on Fri Oct 11 03:40:59 2019

@author: huyn

"""

def findPeakElement(arr):

start ,stop = 0, len(arr)-1

while start+1<stop:

mid = (start+stop)//2

if arr[mid]>arr[mid-1] and arr[mid]>arr[mid+1]:

return mid

elif arr[mid]>arr[mid-1] and arr[mid]<arr[mid+1]:

start = mid

else:

stop = mid

if arr[start]<arr[stop]:

return stop

else:

return start# -\*- coding: utf-8 -\*-

"""

Created on Mon Sep 30 14:50:40 2019

@author: huyn

"""

# given string s, and string t, report all list of subsequence in s that is same as string t

def findIndicesDFS(s,t):

res = []

def dfs(indexS,indexT,path,s,t):

if indexT==len(t):

t = []

for index in path:

t.append(index)

res.append(t)

elif indexT<len(t):

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

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

# print (indexS,i,indexT)

path.append(i)

dfs(i+1,indexT+1,path,s,t)

path.pop()

dfs(0,0,[],s,t)

return res

#s= "abcde"

#t= "bb"

#print (findIndicesDFS(s,t))

def numMatchingSubseq( S, words) :

count = 0

wordDict = {letter:[] for letter in "qwertyuiopasdfghjklzxcvbnm"}

for word in words:

iterator = iter(word)

letter = next(iterator)

if letter not in wordDict:

wordDict[letter]=[]

wordDict[letter].append(iterator)

for letter in S:

currentWordStartWithLetter = wordDict[letter]

# reset this to empty list

wordDict[letter] = []

for iterator in currentWordStartWithLetter:

nextLetter = next(iterator,None)

if nextLetter:

wordDict[nextLetter].append(iterator)

else:

count+=1

return count

S="ricogwqznwxxcpueelcobbbkuvxxrvgyehsudccpsnuxpcqobtvwkuvsubiidjtccoqvuahijyefbpqhbejuisksutsowhufsygtwteiqyligsnbqglqblhpdzzeurtdohdcbjvzgjwylmmoiundjscnlhbrhookmioxqighkxfugpeekgtdofwzemelpyjsdeeppapjoliqlhbrbghqjezzaxuwyrbczodtrhsvnaxhcjiyiphbglyolnswlvtlbmkrsurrcsgdzutwgjofowhryrubnxkahocqjzwwagqidjhwbunvlchojtbvnzdzqpvrazfcxtvhkruvuturdicnucvndigovkzrqiyastqpmfmuouycodvsyjajekhvyjyrydhxkdhffyytldcdlxqbaszbuxsacqwqnhrewhagldzhryzdmmrwnxhaqfezeeabuacyswollycgiowuuudrgzmwnxaezuqlsfvchjfloczlwbefksxsbanrektvibbwxnokzkhndmdhweyeycamjeplecewpnpbshhidnzwopdjuwbecarkgapyjfgmanuavzrxricbgagblomyseyvoeurekqjyljosvbneofjzxtaizjypbcxnbfeibrfjwyjqrisuybfxpvqywqjdlyznmojdhbeomyjqptltpugzceyzenflfnhrptuugyfsghluythksqhmxlmggtcbdddeoincygycdpehteiugqbptyqbvokpwovbnplshnzafunqglnpjvwddvdlmjjyzmwwxzjckmaptilrbfpjxiarmwalhbdjiwbaknvcqovwcqiekzfskpbhgxpyomekqvzpqyirelpadooxjhsyxjkfqavbaoqqvvknqryhotjritrkvdveyapjfsfzenfpuazdrfdofhudqbfnzxnvpluwicurrtshyvevkriudayyysepzqfgqwhgobwyhxltligahroyshfndydvffd"

words = ["iowuuudrgzmw","azfcxtvhkruvuturdicnucvndigovkzrq","ylmmo","maptilrbfpjxiarmwalhbd","oqvuahijyefbpqhbejuisksutsowhufsygtwteiqyligsnbqgl","ytldcdlxqbaszbuxsacqwqnhrewhagldzhr","zeeab","cqie","pvrazfcxtvhkruvuturdicnucvndigovkzrqiya","zxnvpluwicurrtshyvevkriudayyysepzq","wyhxltligahroyshfn","nhrewhagldzhryzdmmrwn","yqbvokpwovbnplshnzafunqglnpjvwddvdlmjjyzmw","nhrptuugyfsghluythksqhmxlmggtcbdd","yligsnbqglqblhpdzzeurtdohdcbjvzgjwylmmoiundjsc","zdrfdofhudqbfnzxnvpluwicurrtshyvevkriudayyysepzq","ncygycdpehteiugqbptyqbvokpwovbnplshnzafun","gdzutwgjofowhryrubnxkahocqjzww","eppapjoliqlhbrbgh","qwhgobwyhxltligahroys","dzutwgjofowhryrubnxkah","rydhxkdhffyytldcdlxqbaszbuxs","tyqbvokpwovbnplshnzafunqglnpjvwddvdlmjjyzmwwxzjc","khvyjyrydhxkdhffyytldcdlxqbasz","jajekhvyjyrydhxkdhffyytldcdlxqbaszbuxsacqwqn","ppapjoliqlhbrbghq","zmwwxzjckmaptilrbfpjxiarm","nxkahocqjzwwagqidjhwbunvlchoj","ybfxpvqywqjdlyznmojdhbeomyjqptltp","udrgzmwnxae","nqglnpjvwddvdlmjjyzmww","swlvtlbmkrsurrcsgdzutwgjofowhryrubn","hudqbfnzxnvpluwicurr","xaezuqlsfvchjf","tvibbwxnokzkhndmdhweyeycamjeplec","olnswlvtlbmkrsurrcsgdzu","qiyastqpmfmuouycodvsyjajekhvyjyrydhxkdhffyyt","eiqyligsnbqglqblhpdzzeurtdohdcbjvzgjwyl","cgiowuuudrgzmwnxaezuqlsfvchjflocz","rxric","cygycdpehteiugqbptyqbvokpwovbnplshnzaf","g","surrcsgd","yzenflfnhrptuugyfsghluythksqh","gdzutwgjofowhryrubnxkahocqjzwwagqid","ddeoincygycdpeh","yznmojdhbeomyjqptltpugzceyzenflfnhrptuug","ejuisks","teiqyligsnbqglqblhpdzzeurtdohdcbjvzgjwylmmoi","mrwnxhaqfezeeabuacyswollycgio","qfskkpfakjretogrokmxemjjbvgmmqrfdxlkfvycwalbdeumav","wjgjhlrpvhqozvvkifhftnfqcfjmmzhtxsoqbeduqmnpvimagq","ibxhtobuolmllbasaxlanjgalgmbjuxmqpadllryaobcucdeqc","ydlddogzvzttizzzjohfsenatvbpngarutztgdqczkzoenbxzv","rmsakibpprdrttycxglfgtjlifznnnlkgjqseguijfctrcahbb","pqquuarnoybphojyoyizhuyjfgwdlzcmkdbdqzatgmabhnpuyh","akposmzwykwrenlcrqwrrvsfqxzohrramdajwzlseguupjfzvd","vyldyqpvmnoemzeyxslcoysqfpvvotenkmehqvopynllvwhxzr","ysyskgrbolixwmffygycvgewxqnxvjsfefpmxrtsqsvpowoctw","oqjgumitldivceezxgoiwjgozfqcnkergctffspdxdbnmvjago","bpfgqhlkvevfazcmpdqakonkudniuobhqzypqlyocjdngltywn","ttucplgotbiceepzfxdebvluioeeitzmesmoxliuwqsftfmvlg","xhkklcwblyjmdyhfscmeffmmerxdioseybombzxjatkkltrvzq","qkvvbrgbzzfhzizulssaxupyqwniqradvkjivedckjrinrlxgi","itjudnlqncbspswkbcwldkwujlshwsgziontsobirsvskmjbrq","nmfgxfeqgqefxqivxtdrxeelsucufkhivijmzgioxioosmdpwx","ihygxkykuczvyokuveuchermxceexajilpkcxjjnwmdbwnxccl","etvcfbmadfxlprevjjnojxwonnnwjnamgrfwohgyhievupsdqd","ngskodiaxeswtqvjaqyulpedaqcchcuktfjlzyvddfeblnczmh","vnmntdvhaxqltluzwwwwrbpqwahebgtmhivtkadczpzabgcjzx","yjqqdvoxxxjbrccoaqqspqlsnxcnderaewsaqpkigtiqoqopth","wdytqvztzbdzffllbxexxughdvetajclynypnzaokqizfxqrjl","yvvwkphuzosvvntckxkmvuflrubigexkivyzzaimkxvqitpixo","lkdgtxmbgsenzmrlccmsunaezbausnsszryztfhjtezssttmsr","idyybesughzyzfdiibylnkkdeatqjjqqjbertrcactapbcarzb","ujiajnirancrfdvrfardygbcnzkqsvujkhcegdfibtcuxzbpds","jjtkmalhmrknaasskjnixzwjgvusbozslrribgazdhaylaxobj","nizuzttgartfxiwcsqchizlxvvnebqdtkmghtcyzjmgyzszwgi","egtvislckyltpfogtvfbtxbsssuwvjcduxjnjuvnqyiykvmrxl","ozvzwalcvaobxbicbwjrububyxlmfcokdxcrkvuehbnokkzala","azhukctuheiwghkalboxfnuofwopsrutamthzyzlzkrlsefwcz","yhvjjzsxlescylsnvmcxzcrrzgfhbsdsvdfcykwifzjcjjbmmu","tspdebnuhrgnmhhuplbzvpkkhfpeilbwkkbgfjiuwrdmkftphk","jvnbeqzaxecwxspuxhrngmvnkvulmgobvsnqyxdplrnnwfhfqq","bcbkgwpfmmqwmzjgmflichzhrjdjxbcescfijfztpxpxvbzjch","bdrkibtxygyicjcfnzigghdekmgoybvfwshxqnjlctcdkiunob","koctqrqvfftflwsvssnokdotgtxalgegscyeotcrvyywmzescq","boigqjvosgxpsnklxdjaxtrhqlyvanuvnpldmoknmzugnubfoa","jjtxbxyazxldpnbxzgslgguvgyevyliywihuqottxuyowrwfar","zqsacrwcysmkfbpzxoaszgqqsvqglnblmxhxtjqmnectaxntvb","izcakfitdhgujdborjuhtwubqcoppsgkqtqoqyswjfldsbfcct","rroiqffqzenlerchkvmjsbmoybisjafcdzgeppyhojoggdlpzq","xwjqfobmmqomhczwufwlesolvmbtvpdxejzslxrvnijhvevxmc","ccrubahioyaxuwzloyhqyluwoknxnydbedenrccljoydfxwaxy","jjoeiuncnvixvhhynaxbkmlurwxcpukredieqlilgkupminjaj","pdbsbjnrqzrbmewmdkqqhcpzielskcazuliiatmvhcaksrusae","nizbnxpqbzsihakkadsbtgxovyuebgtzvrvbowxllkzevktkuu","hklskdbopqjwdrefpgoxaoxzevpdaiubejuaxxbrhzbamdznrr","uccnuegvmkqtagudujuildlwefbyoywypakjrhiibrxdmsspjl","awinuyoppufjxgqvcddleqdhbkmolxqyvsqprnwcoehpturicf"]

print(numMatchingSubseq( S, words))# -\*- coding: utf-8 -\*-

"""

Created on Fri Oct 4 00:46:53 2019

@author: huyn

"""

#Anagram Substring Search (Or Search for all permutations)

#Given a text txt[0..n-1] and a pattern pat[0..m-1], write a function search(char pat[], char txt[])

#that prints all occurrences of pat[] and its permutations (or anagrams) in txt[]. You may assume that n > m.

#Expected time complexity is O(n)

def findAllAnagram(txt,pattern):

res = []

d = {}

currentD = {}

for letter in pattern:

if letter not in d:

d[letter]=0

if letter not in currentD:

currentD[letter] = 0

d[letter]+=1

start,stop = 0,0

length = 0

print (currentD,d)

while stop<len(txt):

print (start,stop,res,currentD)

currentLetter = txt[stop]

if currentLetter not in d:

for i in range(start,stop):

currentD[txt[i]]-=1

start,stop = stop+1,stop+1

length = 0

else:

print ("currentLetter",currentLetter)

if currentD[currentLetter]<d[currentLetter]:

currentD[currentLetter]+=1

length +=1

if length == len(pattern):

res.append(start)

stop+=1

else:

# we currently have currentD[currentLetter] == d[currentLetter]

# we have to move start until we hit the same current letter, or we can't move tstop at all

while start<stop and currentD[currentLetter] == d[currentLetter]:

lastLetter = txt[start]

currentD[lastLetter]-=1

length-=1

start+=1

# eventually, we will have currentD[currentLetter]<d[currentLetter]

currentD[currentLetter]+=1

length+=1

stop+=1

if length == len(pattern):

res.append(start)

return res

txt= "AAABABAA"

pattern ="AABA"

print (findAllAnagram(txt,pattern))# -\*- coding: utf-8 -\*-

"""

Created on Wed Sep 25 20:08:09 2019

@author: huyn

"""

#Find triplets that sum to zero

#Determine if any 3 integers in an array sum to 0.

# just return True False, or the triplet

# find all or just any

# can a number be repeated for the triplets

# any 2 sums make overflow

#O(n^3)

def containsTripletNaive(arr,num):

for i in range(len(arr)):

for j in range(len(arr)):

for k in range(len(arr)):

if arr[i]+arr[j]+arr[k]==num:

return True

return False

#O(n^2) and O(n) space

def containsTripletHash(arr,num):

dictionary = set()

for item in arr:

dictionary.add(item)

for i in range(len(arr)):

for j in range(len(arr)):

mySum = arr[i]+arr[j]

if num-mySum in dictionary:

return True

return False

arr = [-5, -5, -4, -3, -2, -1, 1, 1, 2, 2, 2, 3, 5, 10, 10, 31]

num = 0

#print (containsTripletNaive(arr,num))

#print (containsTripletHash(arr,num))

#O(n^2logn)

def binarySearch(arr,num):

arr.sort()

for i in range(len(arr)):

for j in range(len(arr)):

mySum = arr[i]+arr[j]

target = num-mySum

start,stop =0,len(arr)-1

while start+1<stop:

mid = (start+stop)//2

if arr[mid]==target:

return True

elif arr[mid]<target:

start= mid

else:

stop = mid

if arr[start]==mid:

return True

if arr[stop]==mid:

return True

return False

# 2 pointers, O(n^2)

def pointers(arr,num):

arr.sort()

for i in range(len(arr)):

item = num- arr[i]

start = 0

stop = len(arr)-1

while start<=stop:

s = arr[stop]+arr[start]

if s>item:

stop-=1

elif s<item:

start+=1

else:

return True

return False# -\*- coding: utf-8 -\*-

"""

Created on Thu Oct 10 16:53:28 2019

@author: Huy Nguyen

"""

def flatten(root) -> None:

"""

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

"""

if root:

flatten(root.left)

flatten(root.right)

if root.left and root.right:

temp = root.right

root.right = root.left

root.left = None

while root.right:

root= root.right

root.right = temp

elif root.left:

root.right = root.left

root.left= None# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 9 22:42:47 2019

@author: Huy Nguyen

"""

#Group anagrams

from typing import List

def groupAnagrams(strs: List[str]) -> List[List[str]]:

d = {}

for word in strs:

sortedW = "".join(sorted(word))

if sortedW not in d:

d[sortedW]=[]

d[sortedW].append(word)

return [d[word] for word in d]# -\*- coding: utf-8 -\*-

"""

Created on Tue Sep 17 15:29:56 2019

@author: huyn

"""

#import time

#You have a huge array of integers containing only 0's and 1's. Write an algorithm

# that finds the number of 1's found in the array in a specific range.

def naiveCountOne(arr,start,stop):

count = 0

for i in range(start,stop+1):

if arr[i]:

count+=1

return count

# unmuttable array, we do processing first

def process(arr):

newArr =[]

count = 0

for num in arr:

if num==1:

count+=1

newArr.append(count)

return newArr

def countOneFromProcessedArr(arr,start,stop):

if start ==0:

return arr[stop]

return arr[stop]-arr[start-1]

arr= [0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1]

#start =time.time()

#print (naiveCountOne(arr,3,50))

#stop = time.time()

#print (stop-start)

#arr1= process(arr)

#start =time.time()

#print (countOneFromProcessedArr(arr1,3,50))

#stop = time.time()

#print (stop-start)

#307. Range Sum Query - Mutable

class SegmentNode:

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

self.val = val

self.start = start

self.stop = stop

self.left = left

self.right = right

class NumArray:

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

self.root = self.initialize(nums,0,len(nums)-1)

def initialize(self,nums,start,stop):

if start == stop:

node = SegmentNode(nums[start],start,stop)

return node

elif start<stop:

mid = (start+stop)//2

left = self.initialize(nums,start,mid)

right = self.initialize(nums,mid+1,stop)

node = SegmentNode(left.val+right.val,start,stop,left,right)

node.left = left

node.right = right

return node

def update(self, i: int, val: int) -> None:

def dfs(root,i,val):

if root:

if root.start==root.stop ==i:

extra = val-root.val

root.val = val

return extra

else:

start,stop = root.start,root.stop

# check which half does it belong to, left or right

mid = (start+stop)//2

if i>=start and i<=mid:

extra= dfs(root.left,i,val)

else:

extra = dfs(root.right,i,val)

root.val+=extra

return extra

dfs(self.root,i,val)

def sumRange(self, i: int, j: int) -> int:

def dfs(root,i,j):

start,stop = root.start,root.stop

if i== start and j == stop:

return root.val

else:

mid = (start+stop)//2

val = 0

# check if i>mid

if i>mid:

val+=dfs(root.right,i,j)

elif i<=mid:

if j<=mid:

val+=dfs(root.left,i,j)

else:

# this means we have to search bothway

val+=dfs(root.left,i,mid)

val+=dfs(root.right,mid+1,j)

return val

return dfs(self.root,i,j)

def printOut(self,node):

if node:

self.printOut(node.left)

print (node.val)

self.printOut(node.right)

arr = [1,3,5]

tree = NumArray(arr)

#tree.update(0,10)

#tree.printOut(tree.root)

print (tree.sumRange(0,2))

# -\*- coding: utf-8 -\*-

"""

Created on Mon Sep 23 21:26:07 2019

@author: huyn

"""

#708. Insert into a Cyclic Sorted List

#Given a node from a cyclic linked list which is sorted in ascending order, write a

# function to insert a value into the list such that it remains a cyclic sorted list.

# The given node can be a reference to any single node in the list, and may not be necessarily

# the smallest value in the cyclic list.

#

#If there are multiple suitable places for insertion, you may choose any place to insert

#the new value. After the insertion, the cyclic list should remain sorted.

#

#If the list is empty (i.e., given node is null), you should create a new single cyclic

#list and return the reference to that single node. Otherwise, you should return the original given node.

#

#The following example may help you understand the problem better:

class Node:

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

self.val = val

self.next = next

def insert(self, head: 'Node', insertVal: int) -> 'Node':

root = head

if not head:

root = Node(insertVal)

root.next = root

# minVal = float("inf")

# minNode = None

# maxVal = -float("inf")

# maxNode = Node

# 3 cases

#1 insertVal is less than min

#2 insertVal is greater than max

#3 insertVal <= max and insertVal >=min

# 4 all node are equal, we wont hit differrent between min, max, therefore, have to insert anywhere

else:

head = head.next

while True:

if head.val<=insertVal : # head<val

if head.next.val>=insertVal:# 3 cases, done

newNode = Node(insertVal)

newNode.next = head.next

head.next = newNode

break

else:

# so far looks like second case, we can check if head less than head.next, it means we have a min and max

#head.next <val

if head.val>head.next.val: # we hit max, and min

newNode = Node(insertVal)

newNode.next = head.next

head.next = newNode

break

else:

head = head.next

else: # val <head

if head.next.val>=head.val:

head = head.next # we are in the increasing, wait until we hit the min

else:

if insertVal<head.next.val:

# head.val >head.next.val , we hit a max , min

newNode = Node(insertVal)

newNode.next = head.next

head.next = newNode

break

else:

head= head.next

# run this until we hit root again, this means, we have to insert our insertVal

if head == root:

newNode = Node(insertVal)

newNode.next = head.next

head.next = newNode

break

return root# -\*- coding: utf-8 -\*-

"""

Created on Fri Oct 4 02:02:59 2019

@author: huyn

"""

from typing import List

#Given 2 disjoint sets of intervals, find the intersections

#986. Interval List Intersections

#Given two lists of closed intervals, each list of intervals is pairwise disjoint and in sorted order.

#

#Return the intersection of these two interval lists.

#

#(Formally, a closed interval [a, b] (with a <= b) denotes the set of real numbers

#x with a <= x <= b. The intersection of two closed intervals is a set of real numbers

# that is either empty, or can be represented as a closed interval.

# For example, the intersection of [1, 3] and [2, 4] is [2, 3].)

def intervalIntersection( A: List[List[int]], B: List[List[int]]) -> List[List[int]]:

res = []

i,j = 0,0

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

startA,stopA = A[i]

startB,stopB = B[j]

low = max(startA,startB)

high = min(stopA,stopB)

if low<=high:

res.append([low,high])

if stopA>stopB:

j+=1

else:

i+=1

return res

A = [[0,2],[5,10],[13,23],[24,25]]

B = [[1,5],[8,12],[15,24],[25,26]]

print (intervalIntersection(A,B))# -\*- coding: utf-8 -\*-

"""

Created on Wed Sep 18 16:31:52 2019

@author: Huy Nguyen

"""

#785. Is Graph Bipartite?

#Given an undirected graph, return true if and only if it is bipartite.

#

#Recall that a graph is bipartite if we can split it's set of nodes into two independent subsets A and B such that every

# edge in the graph has one node in A and another node in B.

#

#The graph is given in the following form: graph[i] is a list of indexes j for which the edge between nodes i and j exists.

# Each node is an integer between 0 and graph.length - 1.

# There are no self edges or parallel edges: graph[i] does not contain i, and it doesn't contain any element twice.

# a graph is biPartite if

# we can split it into 2 set(or 2 color code)

def isBipartite(graph):

d = {}

# idea is we break our graph into graph that is connected

# then try to 2 color them

for node,neighbors in enumerate(graph):

if node not in d: # have not assign this node, that means any of the edge was not assigned neither

# we intiate a new stack for this node, preferably keep traversing the stack until stack is done

stack = [node]

d[node] = 0

while stack:

node = stack.pop()

for neighbor in graph[node]:

if neighbor not in d:

# we will explore this node

stack.append(neighbor)

d[neighbor]= 1-d[node]

elif d[neighbor]==d[node]:

# have a clashing if we hit a node with same colore (basically a triangle)

return False

return True

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

print (isBipartite(graph))

#graph = [[1,2,3], [0,2], [0,1,3], [0,2]]

#print (isBipartite(graph))# -\*- coding: utf-8 -\*-

"""

Created on Thu Oct 3 23:44:42 2019

@author: huyn

"""

class TreeNode:

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

self.val = x

self.left = None

self.right = None

#Problem: Is this binary tree a valid binary search tree?

def isValidBSTAssignMinMax(root: TreeNode) -> bool:

if not root:

return True

else:

checkLeft = isValidBSTAssignMinMax(root.left)

checkRight = isValidBSTAssignMinMax(root.right)

if not root.left and not root.right:

root.min = root.val

root.max = root.val

return True

elif root.left and root.right:

leftMin,leftMax = root.left.min, root.left.max

rightMin,rightMax = root.right.min,root.right.max

root.min = min(leftMin,rightMin,root.val)

root.max = max(leftMax,rightMax,root.val)

return root.val>leftMax and root.val<rightMin and checkLeft and checkRight

elif root.left:

leftMin,leftMax = root.left.min, root.left.max

root.min = min(leftMin,root.val)

root.max = max(leftMax,root.val)

return root.val>leftMax and checkLeft and checkRight

else:

rightMin,rightMax = root.right.min,root.right.max

root.min = min(rightMin,root.val)

root.max = max(rightMax,root.val)

return root.val<rightMin and checkLeft and checkRight

def isValidBstPreorder(self, root: TreeNode) -> bool:

self.last = None

def dfs(root):

if not root:

return True

else:

checkL = dfs(root.left)

check = True

if self.last!=None:

if root.val<=self.last:

check= False

self.last = root.val

checkR = dfs(root.right)

return check and checkL and checkR

return dfs(root)

def isValidBstPreorder1(self, root: TreeNode) -> bool:

def dfs(root,path):

if not root:

return True

else:

checkL = dfs(root.left,path)

check = True

last = path[-1]

if last!=None:

if root.val<=last:

check= False

path.pop()

path.append(root.val)

checkR = dfs(root.right,path)

return check and checkL and checkR

return dfs(root,[None])# -\*- coding: utf-8 -\*-

"""

Created on Mon Sep 16 17:25:41 2019

@author: huyn

"""

#An array is monotonic if it is either monotone increasing or monotone decreasing.

#

#An array A is monotone increasing if for all i <= j, A[i] <= A[j]. An array A is monotone

#decreasing if for all i <= j, A[i] >= A[j].

#

#Return true if and only if the given array A is monotonic.

def isMonotonic(A: list[int]) -> bool:

if len(A)==1:

return True

v = A[1]-A[0]

if v ==0:

current = 0

else:

current = abs(v)/v

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

val = A[i+1]-A[i]

if val==0:

continue

val = abs(val)/val

if val!=current and current!=0:

return False

elif current == 0:

current = val

return True

# -\*- coding: utf-8 -\*-

"""

Created on Fri Oct 4 12:20:07 2019

@author: huyn

"""

from typing import List

#973. K Closest Points to Origin

import heapq

def kClosest(points: List[List[int]], K: int) -> List[List[int]]:

myList = []

for x,y in points:

distance = (x\*\*2+y\*\*2)

heapq.heappush(myList,(distance,[x,y]))

return [item[1] for item in heapq.nsmallest(K,myList)]

points = [[3,3],[5,-1],[-2,4]]

K= 2

print (kClosest(points,K))# -\*- coding: utf-8 -\*-

"""

Created on Sat Sep 21 15:39:41 2019

@author: huyn

"""

#340. Longest Substring with At Most K Distinct Characters

#Given a string, find the length of the longest substring T that contains at most k distinct characters.

def lengthOfLongestSubstringKDistinct(s: str, k: int) -> int:

if k == 0:

return 0

start,stop = 0,0

dictionary = {} # this will keep track of how many distinct characters we have and how many of it we have in the current window

length = 0 # keep track of how long was our substring

maxLength = 0 # our output

while stop<len(s):

currentLetter = s[stop]

if currentLetter in dictionary:

dictionary[currentLetter]+=1

length+=1

else:

# 2 cases, if our len(dic) already hit k, this will increase our number of distinct, we need to decrease our unique key before adding this

# we also need to record our length as max if it is greater than maxlength

maxLength =max(length,maxLength)

while len(dictionary)==k: # we dont need to add the start<stop since we know k>=1, we will never have to increase start = stop

lastLetter = s[start]

# decrement the count by 1

dictionary[lastLetter]-=1

length-=1

# pop out the last letter if hits 0

if dictionary[lastLetter]==0:

dictionary.pop(lastLetter)

# it will break the loop after this

start+=1

# now we can add our currentLetter

if currentLetter not in dictionary:

dictionary[currentLetter]=0

dictionary[currentLetter]+=1 # increment our count

length+=1 # increment our length

stop+=1

return max(length,maxLength) # might be a case we dont hit the amount of key, but still greater than maxLength# -\*- coding: utf-8 -\*-

"""

Created on Thu Sep 26 13:57:31 2019

@author: huyn

"""

#Letter Combinations of a Phone Number

#You are given two parameters \* A mapping from digits (represented as strings or chars) to a list of letters.

#You want to return all the possible ways to replace each digit in the input string with its respective letters

#from the mapping.

mapping = {'1': ['A', 'B', 'C'], '2': ['D', 'E', 'F'], '3': ['G', 'H', 'I'], '4': ['J', 'K', 'L'] }

def phone\_permute\_DFS(digits, mapping):

if not digits:

return []

res = []

mapping = {'2': 'ABC', '3': 'DEF', '4': 'GHI', '5': 'JKL' ,

'6':'MNO','7': 'PQRS','8':'TUV','9':'WXYZ'}

def dfs(index,path):

if index==len(digits):

res.append("".join(path))

else:

for letter in mapping[digits[index]]:

path.append(letter.lower())

dfs(index+1,path)

path.pop()

dfs(0,[])

return res

def phone\_permute\_iteration(digits,mapping):

res = [""]

for digit in digits:

temp =[]

for item in res:

for letter in mapping[digit]:

temp.append(item+letter)

res = temp

return res

digits ="123"

comb=phone\_permute\_iteration(digits,mapping)

def verifier(digits,comb):

reverseMap = {letter:key for key in mapping for letter in mapping[key]}

for combination in comb:

if len(combination)!=len(digits):

return False

for i in range(len(digits)):

if reverseMap[combination[i]]!=digits[i]:

return False

count = len(mapping[digits[0]])

for digit in digits[1:]:

count\*=len(mapping[digit])

return count == len(comb)

#print (verifier(digits,comb))# -\*- coding: utf-8 -\*-

"""

Created on Mon Sep 30 12:01:29 2019

@author: huyn

"""

#516. Longest Palindromic Subsequence

#Given a string s, find the longest palindromic subsequence's length in s. You may assume that the maximum length of s is 1000.

s="bbbab"

s="cbbd"

# version used palindrome Substring

def longestPalindromeSubseq(s: str) -> int:

arr =[]

n = len(s)

maxLength = 0

for i in range(n):

t= []

for j in range(n):

if i==j:

t.append(1)

maxLength = 1

else:

t.append(0)

arr.append(t)

for i in range(1,n):

for j in range(1,n):

if i!=j:

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

arr[i][j] = arr[i-1][j-1]+2

else:

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

maxLength = max(maxLength,arr[i][j])

return maxLength

# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 9 16:15:52 2019

@author: huyn

"""

def lengthOfLongestSubstring(s: str) -> int:

mySet = set()

start,stop = 0,0

maxLength =0

currentLength = 0

while stop<len(s):

letter = s[stop]

if letter not in mySet:

currentLength+=1

mySet.add(letter)

else:

maxLength = max(maxLength,currentLength)

while start<stop and s[start]!=letter:

lastLetter = s[start]

currentLength-=1

mySet.remove(lastLetter)

start+=1

if s[start]==letter:

mySet.remove(letter)

currentLength-=1

start+=1

mySet.add(letter)

currentLength+=1

stop+=1

return max(maxLength,currentLength)# -\*- coding: utf-8 -\*-

"""

Created on Sun Sep 29 21:48:40 2019

@author: Huy Nguyen

"""

import random

#longest valid parenthheses substring

def findLongestValid(string):

return

def generateString(n):

string = ""

for i in range(n):

string+=random.choices("()")[0]

return string

string = "()()(((())"

print (findLongestValid(string))

# -\*- coding: utf-8 -\*-

"""

Created on Thu Oct 10 17:44:58 2019

@author: Huy Nguyen

"""

class TreeNode:

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

self.val = x

self.left = None

self.right = None

#Lowest Common Ancestor of a Binary Tree

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

def findPath(root,target,path):

if root:

if root.val == target.val:

path.append(root)

return True

else:

checkLeft = findPath(root.left,target,path)

if checkLeft:

path.append(root)

return True

checkRight = findPath(root.right,target,path)

if checkRight:

path.append(root)

return True

return False

else:

return False

pathP = []

foundP = findPath(root,p,pathP)

pathQ = []

foundQ = findPath(root,q,pathQ)

pathP.reverse()

pathQ.reverse()

for i in range(min(len(pathP),len(pathQ))):

if pathP[i]!=pathQ[i]:

return pathQ[i-1]

return pathQ[i]# -\*- coding: utf-8 -\*-

"""

Created on Fri Oct 11 12:29:17 2019

@author: huyn

"""

#rderedDict is a subclass of dict, and needs more memory to keep track of the order

# in which keys are added. This isn't trivial. The implementation adds a second dict

# under the covers, and a doubly-linked list of all the keys (that's the part that remembers the order),

# and a bunch of weakref proxies. It's not a lot slower, but at least doubles the memory over using a

# plain dict.

#

#But if it's appropriate, use it! That's why it's there :-)

# But if this were a Python list, deleting a key would take O(n) time twice over:

#O(n) time to find the key in the list, and O(n) time to remove the key from the list.

#So it's a doubly-linked list instea

from collections import OrderedDict

class LRUCache(OrderedDict):

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

"""

:type capacity: int

"""

self.capacity = capacity

def get(self, key):

"""

:type key: int

:rtype: int

"""

if key not in self:

return - 1

self.move\_to\_end(key) # we use this, so we make it most recent again

return self[key]

def put(self, key, value):

"""

:type key: int

:type value: int

:rtype: void

"""

if key in self:

self.move\_to\_end(key)

self[key] = value

if len(self) > self.capacity:

self.popitem(last = False) # pop the least used, which is order the first# -\*- coding: utf-8 -\*-

"""

Created on Mon Sep 23 23:18:04 2019

@author: huyn

"""

#124. Binary Tree Maximum Path Sum

#Given a non-empty binary tree, find the maximum path sum.

#

#For this problem, a path is defined as any sequence of nodes from some starting

#node to any node in the tree along the parent-child connections. The path must

#contain at least one node and does not need to go through the root.

class TreeNode:

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

self.val = x

self.left = None

self.right = None

def maxPathSum(self,root: TreeNode) -> int:

self.max = -float("inf")

def dfs(node):

if not node:

return 0

else:

left = max(dfs(node.left),0)

right = max(dfs(node.right),0)

throughNode = node.val+left+right

self.max = max(self.max,throughNode)

return node.val+max(left,right)

dfs(root)

return self.max# -\*- coding: utf-8 -\*-

"""

Created on Thu Oct 10 15:00:39 2019

@author: Huy Nguyen

"""

# merge 2 list given 2 list node

def mergeTwoLists(l1: ListNode, l2: ListNode) -> ListNode:

head = ListNode(None)

def dfs(l1,l2,currentHead):

if l1 and l2:

if l1.val>l2.val:

currentHead.next= l2

currentHead = l2

dfs(l1,l2.next,currentHead)

else:

currentHead.next = l1

currentHead = l1

dfs(l1.next,l2,currentHead)

elif l1:

currentHead.next=l1

elif l2:

currentHead.next = l2

dfs(l1,l2,head)

return head.next# -\*- coding: utf-8 -\*-

"""

Created on Wed Sep 25 22:34:49 2019

@author: huyn

"""

#Find Intersection of Two Sorted Integer Arrays

def mergerPointers(arr1,arr2):

p1,p2 = 0

arr = []

while p1<len(arr1) and p2<len(arr2):

num1,num2= arr1[p1],arr2[p2]

if num1==num2:

arr.append(num1)

arr.append(num2)

p1+=1

p2+=1

elif num1<num2:

arr.append(num1)

p1+=1

else:

arr.append(num2)

p2+=1

for i in range(p1,len(arr1)):

arr.append(arr1[i])

for i in range(p2,len(arr2)):

arr.append(arr2[i])

return arr

arr1,arr2=[1,3,6,17,21], [2,4,6,21]

# -\*- coding: utf-8 -\*-

"""

Created on Wed Sep 18 22:19:13 2019

@author: huyn

"""

#56. Merge Intervals

#Given a collection of intervals, merge all overlapping intervals.

def merge(intervals):

if len(intervals)<=1:

return intervals

# sort the interval

intervals.sort()

currentInterval = intervals[0]

output = []

for interval in intervals[1:]:

currentStart,currentStop = currentInterval

start,stop = interval

# if current interval start less or equal to stop

if currentStart<=stop:

# for case where they dont overlap [1,3], [5,6]

# we can also can check max(start,currentstart)>min(stop,currentStop) for overlap

if currentStop<start:

output.append(currentInterval)

currentInterval = interval # update our currentInterval

else:

# our new interval get to merge

currentInterval = [min(start,currentStart),max(currentStop,stop)]

# print (currentInterval)

output.append(currentInterval) # append the current one

return output

intervals = [[1,3],[2,6],[8,10],[15,18]]

print ("output:",merge(intervals))

intervals = [[1,4],[4,5]]

print (merge(intervals))# -\*- coding: utf-8 -\*-

"""

Created on Sun Sep 22 17:29:05 2019

@author: huyn

"""

from typing import List

import heapq

#23. Merge k Sorted Lists

#Merge k sorted linked lists and return it as one sorted list. Analyze and describe its complexity.

class ListNode:

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

self.val = x

self.next = None

def mergeKListsV1(self, lists: List[ListNode]) -> ListNode:

root = ListNode(0) # this is our root

head = root

while True:

currentMin = float("inf")

i = None

for index in range(len(lists)):

currentNode = lists[index]

if currentNode:

val = currentNode.val

if val <currentMin:

val = currentMin

i = index # store this so we can do node.next

if i == None: # this means we have gone through all the node

break

else:

# create new node

newNode = ListNode(currentMin)

head.next = newNode

head = head.next

# make the node at array index i become next

lists[i] = lists[i].next

return root.next

def mergeKListsV2(self, lists: List[ListNode]) -> ListNode:

h = [(l.val, idx) for idx, l in enumerate(lists) if l]

heapq.heapify(h) # make it a mean heap by value

head = cur = ListNode(None)

while h:

val, idx = heapq.heappop(h)

cur.next = ListNode(val)

cur = cur.next

node = lists[idx] = lists[idx].next

if node:

heapq.heappush(h, (node.val, idx))

return head.next# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 9 23:45:02 2019

@author: Huy Nguyen

"""

#merge sorted array in place

def merge(self, nums1, m, nums2, n):

"""

:type nums1: List[int]

:type m: int

:type nums2: List[int]

:type n: int

:rtype: void Do not return anything, modify nums1 in-place instead.

"""

p1 = m-1

p2 = n-1

while p1>=0 and p2>=0:

n1 = nums1[p1]

n2 = nums2[p2]

if n2>=n1:

nums1[p1+p2+1]= n2

p2-=1

elif n2<n1:

nums1[p1+p2+1]= n1

p1-=1

while p2>=0:

nums1[p2]=nums2[p2]

p2-=1# -\*- coding: utf-8 -\*-

"""

Created on Sun Sep 29 21:47:22 2019

@author: Huy Nguyen

"""

import random

#921. Minimum Add to Make Parentheses Valid

#Given a string S of '(' and ')' parentheses, we add the minimum number of parentheses ( '(' or ')', and in any positions ) so t

#hat the resulting parentheses string is valid.

#

#Formally, a parentheses string is valid if and only if:

#

#It is the empty string, or

#It can be written as AB (A concatenated with B), where A and B are valid strings, or

#It can be written as (A), where A is a valid string.

#Given a parentheses string, return the minimum number of parentheses we must add to make the resulting string valid.

def minAddToMakeValid(self, S: str) -> int:

countL = 0

res = 0

for item in S:

if item =="(":

countL+=1

else:

if countL==0:

res+=1

else:

countL-=1

return res+countL

def generateString(n):

string = ""

for i in range(n):

string+=random.choices("()")[0]

return string

string =generateString(100)

print (string)

print (minAddToMakeValid(string))# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 9 22:57:26 2019

@author: Huy Nguyen

"""

#Minimum Window Substring

#Given a string S and a string T, find the minimum window in S which will contain all the characters in T in complexity O(n).

def minWindow(str1: str, str2: str) -> str:

if str2 in str1:

return str2

d = {}

for l in str2:

if l not in d:

d[l]=0

d[l]+=1

# number of different character, use this to mark if we already match str2

hitting = 0

start,stop = 0,0

currentD = {}

output = ""

while stop<len(str1):

letter = str1[stop]

if letter not in currentD:

currentD[letter]=0

currentD[letter]+=1

if letter in d and d[letter]==currentD[letter]:

hitting+=1

# print (currentD,hitting)

# if we hit the form, we will retract just like our template, we retract as long as our hitting is still equal to len(d)

while hitting ==len(d) and start<stop:

# print ("inner loop")

# we check if our current is the smallest string

if not output:

output = str1[start:stop+1]

# print (84,output)

else:

if (stop-start+1)<len(output):

output = str1[start:stop+1]

# print (88,output)

# get the character on start

leftMostLetter =str1[start]

# print ("leftMostLetter",leftMostLetter)

# decrease our count

currentD[leftMostLetter]-=1

# print (currentD)

if leftMostLetter in d and currentD[leftMostLetter]<d[leftMostLetter]:

hitting-=1

# increment our start

start+=1

# increment our stop

stop+=1

return output# -\*- coding: utf-8 -\*-

"""

Created on Wed Sep 18 21:57:25 2019

@author: huyn

"""

import random,heapq

#253. Meeting Rooms II

#Given an array of meeting time intervals consisting of start and end times

#[[s1,e1],[s2,e2],...] (si < ei), find the minimum number of conference rooms required.\

# O(N) space and O(N^2) times

def minMeetingRoomsSlow(intervals):

if len(intervals)<=1:

return len(intervals)

intervals.sort()

count =0

d = {0: [intervals[0]]}

for start,stop in intervals[1:]:

# we check to see if we can add to the last of the list of a current room number

currentMin = float("inf")

roomNum = None

found = False

for num in range(count+1):

lastStart,lastStop = d[num][-1]

if start<lastStop:

continue

elif start == lastStop:

d[num].append([start,stop])

found = True

roomNum = None

break

else:

val = start-lastStop

if val<currentMin:

currentMin= val

roomNum = num

found = True

# if roomNum == None means that dont find any room can store the start stop, initiate new

if not found:

count+=1

d[count]= [[start,stop]]

else:

if roomNum!=None:

d[roomNum].append([start,stop])

return count+1

def generateTest(n):

arr = []

for i in range(n):

start = random.randint(0,30)

stop = random.randint(start+1,start+30)

arr.append([start,stop])

return arr

#intervals = [[0, 30],[5, 10],[15, 20]]

#print (minMeetingRooms(intervals))

#intervals = [[7,10],[2,4]]

#print (minMeetingRooms(intervals))

#intervals = generateTest(10000)

#print (intervals)

#intervals = [[1, 28], [1, 29], [13, 28], [16, 29], [23, 31], [25, 45], [26, 48], [29, 44], [29, 54], [30, 34]]

#print (minMeetingRooms(intervals))

def minMeetingRoomsMinHeap(intervals):

myList = []

intervals.sort()

heapq.heappush(myList,intervals[0][1])

count = 1

for start,stop in intervals[1:]:

currentMin = myList[0]

if start>=currentMin:

heapq.heappop(myList)

heapq.heappush(myList,stop)

else:

# create new room

count+=1

heapq.heappush(myList,stop)

return count

#intervals = [[1, 28], [1, 29], [13, 28], [16, 29], [23, 31], [25, 45], [26, 48], [29, 44], [29, 54], [30, 34]]

intervals = []

#print (minMeetingRoomsMinHeap(intervals))

def minMeetingByOverlap(intervals):

dictionary = {}

for start,stop in intervals:

if start not in dictionary:

dictionary[start]= 0

dictionary[start]+=1

if stop not in dictionary:

dictionary[stop]= 0

dictionary[stop]-=1

myList = sorted(dictionary)

currentRoom, minRoom = 0,0

for time in myList:

currentRoom+=dictionary[time]

minRoom= max(minRoom,currentRoom)

return minRoom

#print (minMeetingByOverlap(intervals))

# -\*- coding: utf-8 -\*-

"""

Created on Thu Oct 3 20:39:17 2019

@author: huyn

"""

#https://www.geeksforgeeks.org/minimum-number-platforms-required-railwaybus-station/

#Minimum Number of Platforms Required for a Railway/Bus Station

#Given arrival and departure times of all trains that reach a railway station,

#the task is to find the minimum number of platforms required for the railway station so that no train waits.

#We are given two arrays which represent arrival and departure times of trains that stop

def minPlatform(arrival,departure):

count = 0

return count# -\*- coding: utf-8 -\*-

"""

Created on Mon Oct 7 12:02:38 2019

@author: huyn

"""

#283. Move Zeroes

#

#Given an array nums, write a function to move all 0's to the end of it while

#maintaining the relative order of the non-zero elements.

from typing import List

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

start, stop = 0,0 #

while stop<len(nums):

if nums[stop]!=0: # we hit a non zero number, check if there is any zeroes before that

while nums[start]!=0 and start<stop: # we scan the start to hit the first 0 and start still less or equal to stop

start+=1

# the loop break once nums[start] =0 and start still less than stop or start is equal to stop

if nums[start]== 0:

# swap

nums[start],nums[stop]= nums[stop],nums[start]

start+=1

stop+=1

nums =[0,1,0,3,12]

moveZeroes(nums)

print (nums)

# -\*- coding: utf-8 -\*-

"""

Created on Mon Sep 16 17:33:11 2019

@author: huyn

"""

import time

#311. Sparse Matrix Multiplication

#Given two sparse matrices A and B, return the result of AB.

#

#You may assume that A's column number is equal to B's row number.

def multiply( A, B):

row = len(A)

col = len(B[0])

output = []

d ={"R":set(),"C":set()}

for r in range(row):

temp = []

for c in range(col):

if r in d["R"]:

temp = [0]\*len(A[0])

continue

else:

val = 0

if c not in d["C"]:

val = 0

# do this to check whether to indicate whole row in A is 0 or whole col in B is 0

checkRow= False

checkCol= False

for k in range(len(A[0])):

if A[r][k]!=0 and B[k][c]!=0:

val+=A[r][k]\*B[k][c]

if A[r][k]!=0:

checkRow= True

if B[k][c]!=0:

checkCol= True

if not checkRow:

d["R"].add(r)

if not checkCol:

d["C"].add(c)

temp.append(val)

output.append(temp)

return output

def multiplyNormal( A, B):

row = len(A)

col = len(B[0])

output = []

for r in range(row):

temp = []

for c in range(col):

val = 0

# for k in range()

for k in range(len(A[0])):

val+=A[r][k]\*B[k][c]

temp.append(val)

output.append(temp)

return output

A = [

[ 1, 0, 0,1],

[-1, 0, 3,1],

[0,0,0,0],

[1,-1,10,2]

]

B = [

[ 7, 0, 0 ,0],

[ 0, 0, 0 ,0],

[ 0, 0, 1,0 ],

[1,2,3,0]

]

#start = time.time()

#print (multiplyNormal(A,B))

#stop = time.time()

#print ((stop-start))

#start = time.time()

#print (multiply(A,B))

#stop = time.time()

#print ((stop-start))# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 9 22:08:47 2019

@author: Huy Nguyen

"""

#Multiply Strings

#Given two non-negative integers num1 and num2 represented as strings, return the product of num1 and num2, also represented as a string.

def multiply(self, num1: str, num2: str) -> str:

if len(num2)>len(num1):

num1,num2=num2,num1

num1 = [int(item) for item in num1]

num2 = [int(item) for item in num2]

listToAdd= []

for i in range(len(num2)-1,-1,-1):

over= 0

string = [0]\*(len(num2)-i-1)

n1 = num2[i]

if n1==0:

continue

for j in range(len(num1)-1,-1,-1):

val = num2[i]\*num1[j]+over

string.append(val%10)

over = val//10

if over:

string.append(over)

listToAdd.append(string)

return addAll(listToAdd)

def addAll(listToAdd):

if len(listToAdd)==0:

return "0"

elif len(listToAdd)==1:

return "".join([str(item) for item in listToAdd[0][::-1]])

else:

first = add2List(listToAdd[0],listToAdd[1])

for eachList in listToAdd[2:]:

second = add2List(first,eachList)

first = second

return "".join([str(item) for item in first[::-1]])

def add2List(list1,list2):

res =[]

over = 0

for i in range(min(len(list1),len(list2))):

n1 = list1[i]

n2 = list2[i]

val = n1+n2+over

res.append(val%10)

over = val//10

for item in list1[i+1:]:

val = item+over

res.append(val%10)

over = val//10

for item in list2[i+1:]:

val = item+over

res.append(val%10)

over = val//10

if over:

res.append(over)

return res# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 9 22:08:47 2019

@author: Huy Nguyen

"""

#Multiply Strings

#Given two non-negative integers num1 and num2 represented as strings, return the product of num1 and num2, also represented as a string.

def multiply(self, num1: str, num2: str) -> str:

if len(num2)>len(num1):

num1,num2=num2,num1

num1 = [int(item) for item in num1]

num2 = [int(item) for item in num2]

listToAdd= []

for i in range(len(num2)-1,-1,-1):

over= 0

string = [0]\*(len(num2)-i-1)

n1 = num2[i]

if n1==0:

continue

for j in range(len(num1)-1,-1,-1):

val = num2[i]\*num1[j]+over

string.append(val%10)

over = val//10

if over:

string.append(over)

listToAdd.append(string)

return addAll(listToAdd)

def addAll(listToAdd):

if len(listToAdd)==0:

return "0"

elif len(listToAdd)==1:

return "".join([str(item) for item in listToAdd[0][::-1]])

else:

first = add2List(listToAdd[0],listToAdd[1])

for eachList in listToAdd[2:]:

second = add2List(first,eachList)

first = second

return "".join([str(item) for item in first[::-1]])

def add2List(list1,list2):

res =[]

over = 0

for i in range(min(len(list1),len(list2))):

n1 = list1[i]

n2 = list2[i]

val = n1+n2+over

res.append(val%10)

over = val//10

for item in list1[i+1:]:

val = item+over

res.append(val%10)

over = val//10

for item in list2[i+1:]:

val = item+over

res.append(val%10)

over = val//10

if over:

res.append(over)

return res# -\*- coding: utf-8 -\*-

"""

Created on Thu Sep 19 21:29:47 2019

@author: Huy Nguyen

"""

#import random

#31. Next Permutation

#Implement next permutation, which rearranges numbers into the lexicographically next greater permutation of numbers.

#

#If such arrangement is not possible, it must rearrange it as the lowest possible order (ie, sorted in ascending order).

#

#The replacement must be in-place and use only constant extra memory.

#

#Here are some examples. Inputs are in the left-hand column and its corresponding outputs are in the right-hand column.

def nextPermutation(nums):

for i in range(len(nums)-1,0,-1):

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

# found the place to change

# going back from this end, check the number to the right of i so that

# it is greater than nums[i-1], the right most, and the minimum

rightmostIndex = None

currentMin = float("inf")

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

if nums[j]>nums[i-1] and currentMin>nums[j] :

rightmostIndex= j

currentMin = nums[j]

# print (rightmostIndex,nums[rightmostIndex],i-1,nums[i-1])

temp = nums[i-1]

nums[i-1]= nums[rightmostIndex]

nums[rightmostIndex]=temp

nums[i:]=sorted(nums[i:])

return nums

nums.sort()

return nums

#for j in range(20):

# print ([int(i) for i in str(random.randint(1,10000000000000000000000000000000000000000000))])

# -\*- coding: utf-8 -\*-

"""

Created on Wed Sep 25 19:56:22 2019

@author: huyn

"""

from collections import deque

#Count all the islands in a binary matrix

#You are given a 2D binary matrix as an input. You want to return the number of

#islands in the binary matrix. You can think of the 0's as the ocean and the 1's as land.

#An island is surrounded by water and is formed by connecting adjacent lands horizontally

#or vertically. You goal is to return the correct number of islands.

def countIslandDFS(grid):

if not grid:

return 0

row = len(grid)

col = len(grid[0])

visited = [[False]\*col for i in range(row)]

numIsland = 0 # count the number of island

# we dfs through our island, and add node to visited

def dfs(currentRow,currentCol,row,col):

# set our directions (4 direction)

directions = [[0,1],[1,0],[0,-1],[-1,0]]

for addRow,addCol in directions:

newRow,newCol = currentRow+addRow,currentCol+addCol

# we check if newRow,newCol valid and has not been visited

if newRow>=0 and newCol>=0 and newCol<col and newRow<row:

if not visited[newRow][newCol] and grid[newRow][newCol]=="1":

# add to visited

visited[newRow][newCol] = True

dfs(newRow,newCol,row,col)

for i in range(row):

for j in range(col):

if grid[i][j]=="1" and not visited[i][j]:

# increment island

numIsland+=1

visited[i][j]= True

dfs(i,j,row,col)

return numIsland

matrix = [[1, 1, 0, 0, 0],

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

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

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

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

#print (countIslandDFS(matrix))

def countIslandBFS(matrix):

row = len(matrix)

col = len(matrix[0])

visited = [[False]\*col for i in range(row)]

numIsland = 0

for r in range(row):

for c in range(col):

if matrix[r][c] and not visited[r][c]:

numIsland+=1

visited[r][c]=True

queue = deque([[r,c]])

while queue:

size = len(queue)

for i in range(size):

currentRow,currentCol = queue.popleft()

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

for addRow,addCol in direction:

newRow,newCol= currentRow+addRow,currentCol+addCol

if newRow>=0 and newCol>=0 and newRow<row and newCol<col and not visited[newRow][newCol] and matrix[newRow][newCol]:

visited[newRow][newCol] = True

queue.append([newRow,newCol])

return numIsland

#print (countIslandBFS(matrix))# -\*- coding: utf-8 -\*-

"""

Created on Sun Sep 22 23:28:22 2019

@author: huyn

"""

import random

#273. Integer to English Words

#Convert a non-negative integer to its english words representation. Given input is guaranteed to be less than 231 - 1.

def numberToWords(num):

if not num:

return "Zero"

# idea is to break into single digits, two digits and then combine every 3 number

single\_digits = ["", "One", "Two", "Three",

"Four", "Five", "Six", "Seven",

"Eight", "Nine"]

# The first string is not used,

# it is to make array indexing simple

two\_digits = ["Ten", "Eleven", "Twelve",

"Thirteen", "Fourteen", "Fifteen",

"Sixteen", "Seventeen", "Eighteen",

"Nineteen"]

# The first two string are not used,

# they are to make array indexing simple

tens\_multiple = ["Twenty", "Thirty", "Forty",

"Fifty", "Sixty", "Seventy", "Eighty",

"Ninety"]

tens\_power = ["", "Thousand","Million","Billion"]

hundred = "Hundred" # special case that reach to 3 number

myList = [] # my main result that store every 3 number

index = 0 # this index indicate in which ten powers I am at

single = [] # store at most 3 number

while num:

val = tens\_power[index]

if val: # check if we have reach thousand mark or above

single.append(val)

# we will take 3 number from right, each time

# we take first digit

firstDigit = num%10

num = num//10

val = single\_digits[firstDigit]

# we see if we can retrive second digit

if num:

secondDigit = num%10

num = num//10

# we start binding our two number, however we need to check if 00, or 0 case

# if secondigit is 0, we wont add any of our ten multiple

if secondDigit==0:

# check if our val is not 0, then we have to append the signe digit first

if val:

single.append(val)

# special cases for two digits

elif secondDigit==1:

# if it is 1 on the two digit, then we can query the index by the value of our first digit 12->2->twelve

single.append(two\_digits[firstDigit])

else:

# if it is not 0, or 1, then we can append firstly our val of not 0

if val:

single.append(val)

# and then append our secondigit

single.append(tens\_multiple[secondDigit-2]) # or i can set tens\_multiple = ["","","Twenty", "Thirty", "Forty",

# "Fifty", "Sixty", "Seventy", "Eighty",

# "Ninety"]

else:

# only 1 digit

if val:

single.append(val)

# check if we can pull the third digit

if num:

thirdDigit = num%10

num = num //10

# have 3 digit

# if there is a non 0 at thrird digit, we need the hundred thingy

if thirdDigit:

single.append(hundred)

val = single\_digits[thirdDigit]

# add the string that correspond to our digit at third place if not 0

if val:

single.append(val)

index+=1

# there could be a case 000 1, that we should not extedn our single into my list or for cases that we actually have 1 number (1 -> One)

if len(single)>1 or (len(single)==1 and single[0] not in tens\_power):# first len(single)>1 ->1 000 000, single = ["thounsand","milluon one"]

myList.extend(single)

single = []

# print (myList)

return " ".join(myList[::-1])

#for i in range(100):

# print (random.randint(0,2\*\*10))# -\*- coding: utf-8 -\*-

"""

Created on Thu Sep 19 12:31:26 2019

@author: huyn

"""

#Write a function to return if two words are exactly "one edit" away, where an edit is:

#Inserting one character anywhere in the word (including at the beginning and end)

#Removing one character

#Replacing exactly one character

def isOneEditAway(w1,w2):

if abs(len(w1)-len(w2))>=2:

return False

if len(w1)==len(w2):

count = 0

for i in range(len(w1)):

if w1[i]!=w2[i]:

if count ==1:

return False

elif count==0:

count+=1

return count==1

else:

i,j=0,0

while i<len(w1) and j<len(w2):

char1= w1[i]

char2= w2[j]

if char1==char2:

i+=1

j+=1

else:

return w1[i+1:]==w2[j:] or w2[j+1:]==w1[i:]

return True # for case that is 1 empty and 1 of length 1, not going through the loop

print (isOneEditAway("abcmef","abcef"))

print (isOneEditAway("cat", "dog"))

print (isOneEditAway("cat", "cats"))

print (isOneEditAway("cat", "cut"))

print (isOneEditAway("cat", "cast"))

print (isOneEditAway("cat", "at"))

print (isOneEditAway("", "a"))# -\*- coding: utf-8 -\*-

"""

Created on Mon Sep 30 12:21:40 2019

@author: huyn

"""

import random

#647. Palindromic Substrings

#Given a string, your task is to count how many palindromic substrings in this string.

#

#The substrings with different start indexes or end indexes are counted as different

#substrings even they consist of same characters.

def countSubstrings(s: str) -> int:

return

def generateString(n):

string = ""

for i in range(n):

string+=random.choice("asd")[0]

return string

s="aaa"

string = generateString(10)# -\*- coding: utf-8 -\*-

"""

Created on Sun Oct 6 19:26:39 2019

@author: huyn

"""

#416. Partition Equal Subset Sum

#Given a non-empty array containing only positive integers, find if the array can

# be partitioned into two subsets such that the sum of elements in both subsets is equal.

from typing import List

def canPartition(arr: List[int]) -> bool:

s = sum(arr)

if s%2:

return False

K = s//2

row = len(arr)+1

col = K+1

dp = [[False]\*col for i in range(row)]

for r in range(row):

dp[r][0]=True

for r in range(1,row):

for c in range(1,col):

dp[r][c] = dp[r-1][c] or dp[r-1][c-arr[r-1]]

return dp[row-1][col-1]

arr=[1, 5, 11, 5]

print (canPartition(arr))

arr=[1, 2, 3, 5]

print (canPartition(arr))# -\*- coding: utf-8 -\*-

"""

Created on Fri Oct 11 01:22:55 2019

@author: huyn

"""

#Permutations II

def permuteUnique(nums):

res = set()

def dfs(index):

if index == len(nums):

res.add(tuple(nums[:]))

else:

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

# we swap the index, we start at i == index so we keep the very first initial, no swapping

# only swap if 2 number are different:

if nums[i]!=nums[index] or i==index:

nums[index],nums[i] = nums[i],nums[index]

dfs(index+1)

nums[i],nums[index] = nums[index],nums[i]

dfs(0)

return [list(item) for item in res]

print (permuteUnique([1,1,2]))# -\*- coding: utf-8 -\*-

"""

Created on Fri Oct 11 01:04:13 2019

@author: huyn

"""

#Permutations

def permute(nums):

res = []

def dfs(path):

if len(nums)==0:

res.append(path[:]) # deep copu

else:

for i in range(len(nums)):

number = nums.pop(i)

path.append(number)

dfs(path)

path.pop()

nums.insert(i,number)

dfs([])

return res

def permuteSwap(nums):

res = []

def dfs(index):

if index == len(nums):

res.append(nums[:])

else:

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

# we swap the index, we start at i == index so we keep the very first initial, no swapping

nums[index],nums[i] = nums[i],nums[index]

dfs(index+1)

nums[i],nums[index] = nums[index],nums[i]

dfs(0)

return res

print (permuteSwap([1,2,3]))# -\*- coding: utf-8 -\*-

"""

Created on Thu Oct 3 21:46:45 2019

@author: huyn

"""

#Point in max overlapping intervals

#Given number M and N intervals in the form [a, b] (inclusive) where for every

# interval -M <= a <= b <= M, create a program that returns a point where the maximum number of intervals overlap.

input =[[0,2],[0,2],[2,4],[3,8],[5,10]]

def find(M,intervals):

d = {}

for i in range(-M,M+1):

d[i]=0

for start,stop in intervals:

d[start]+=1

d[stop]-=1

currentOverlap, currentMax = 0,0

for key in range(-M,M+1):

currentOverlap+=d[key]

if currentOverlap>currentMax:

output = key

currentMax = currentOverlap

return output

print (find(10,input))

# if one end at time t1, and another start at time t1, the end at t1 does not account overlap

def findAllNotCountingTouch(M,intervals):

d = {}

for i in range(-M,M+1):

d[i]=0

for start,stop in intervals:

d[start]+=1

d[stop]-=1

currentOverlap, currentMax = 0,0

arr = []

for key in range(-M,M+1):

# when hitting a key that has more stop than end, and our currentOverlap was equal to currentMax

# this indicate our max interval coming to an end

if d[key]<0 and currentOverlap==currentMax:

arr.append([start,key])

# reset start

start = None

# if our currentOverlap == currentMax -1

currentOverlap+=d[key]

if currentOverlap>currentMax:

# reset our array

arr =[]

start = key

currentMax = currentOverlap

# if we start hitting our max again

elif currentOverlap==currentMax and start == None:

start = key

stop = key

return arr

#print (findAllNotCountingTouch(10,input))

# if one end at time t1, and another start at time t1, the end at t1 does account for overlap

def findAllCountingTouch(M,intervals):

d = {}

for i in range(-M,M+1):

d[i]=[]

for start,stop in intervals:

d[start].append(1)

d[stop].append(-1)

currentOverlap, currentMax = 0,0

arr = []

for key in range(-M,M+1):

# check len of our d[key]

if len(d[key])>currentMax:

# if it combines of stop and start, then it is just a small t1,t1

currentMax = len(d[key])

if -1 in d[key]:

arr= [[key,key]]

else:

start = key

val = sum(d[key])

# when hitting a key that has more stop than end, and our currentOverlap was equal to currentMax

# this indicate our max interval coming to an end

if val<0 and currentOverlap==currentMax:

arr.append([start,key])

# reset start

start = None

# if our currentOverlap == currentMax -1

currentOverlap+=val

if currentOverlap>currentMax:

# reset our array

arr =[]

start = key

currentMax = currentOverlap

# if we start hitting our max again

elif currentOverlap==currentMax and start == None:

start = key

stop = key

return arr

#print (findAllCountingTouch(10,input))# -\*- coding: utf-8 -\*-

"""

Created on Fri Oct 11 02:13:16 2019

@author: huyn

"""

# power to

def myPow(x,n):

if n==0:

return 1

if x== 0:

return 0

if x==1 or x==-1:

if n%2==0:

return 1

else:

return -1

if x<0:

return -myPow(x,n)

if n <0:

return 1/(myPow(x,-n))

if n%2==0:

return myPow(x,n//2)\*myPow(x,n//2)

else:

return myPow(x,n//2)\*myPow(x,n//2)\*x# -\*- coding: utf-8 -\*-

"""

Created on Tue Oct 8 16:59:47 2019

@author: huyn

"""

# Prime String

# Given a string consisting of English letters both lowercase and uppercase.

# Your task is to convert it to the prime word. Prime word is a word consisting

# of only prime characters and prime character is a letter whose ASCII code is prime.

# Each non-prime character is equidistant with 2 prime characters the one with lower

# ASCII value will be consider as its replacement.

#

#Constraints

#1 <= string length <= 5000

#

#Example

#Input: ABc

#Output: CCa

def isPrime(val):

for i in range(2,int(val\*\*.5)):

if val%i==0:

return False

return True

def getPrimeString(string):

letter="qwertyuiopasdfghjklzxcvbnm"

upperLetter = letter.upper()

letter +=upperLetter

letter = sorted(letter)

d = {}

v = {}

res = ""

notAssigned = []

for l in letter:

val = ord(l)

if isPrime(val):

d[l] = l

v[val]=l

else:

notAssigned.append(l)

choices = sorted(v)

for letter in notAssigned:

val =ord(letter)

start,stop =0,len(choices)-1

while start+1<stop:

mid = (start+stop)//2

if choices[mid]>val:

stop=mid

else:

start=mid

if choices[start]>val:

d[letter] = v[choices[start]]

elif choices[stop]<val:

d[letter] = v[choices[stop]]

elif val- choices[start]<=choices[stop]-val:

d[letter] = v[choices[start]]

else:

d[letter] = v[choices[stop]]

# print (d)

for s in string:

res+=d[s]

return res

a= "ABc"

#print (getPrimeString(a))

# -\*- coding: utf-8 -\*-

"""

Created on Thu Oct 10 11:48:40 2019

@author: Huy Nguyen

"""

#Product of Array Except Self

from typing import List

def productExceptSelf(nums: List[int]) -> List[int]:

# consider array [1,2,3,4,5]

# accumulate from left [1,1,1\*2,1\*2\*3,1\*2\*3\*4]

# accumulate from right [2\*3\*4\*5,3\*4\*5,4\*5,5,1]

left = [1]

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

left.append(left[-1]\*nums[i])

right = [1]

for i in range(len(nums)-1,0,-1):

right.append(right[-1]\*nums[i])

res = []

for i in range(len(nums)):

j = len(nums)-i-1

res.append(left[i]\*right[j])

return res# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 9 15:34:49 2019

@author: huyn

"""

import random

#528. Random Pick with Weight

#Given an array w of positive integers, where w[i] describes the weight of index i,

# write a function pickIndex which randomly picks an index in proportion to its weight.

class randomlyWeight:

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

self.w= w

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

self.w[i]+=self.w[i-1]

self.max = w[-1]

def pickIndex(self) -> int:

num = random.randint(0,self.max)

# binary search

start,stop = 0, len(self.w)-1

while start+1<stop:

mid = (start+stop)//2

if self.w[mid]<num:

start = mid

elif self.w[mid]>num:

stop = mid

else:

return mid

if self.w[start]>=num:

return start

if self.w[stop]>=num:

return stop

else:

print (start,stop,num)

myWeights = randomlyWeight([1,3])

print (myWeights.pickIndex())# -\*- coding: utf-8 -\*-

"""

Created on Thu Sep 19 15:42:27 2019

@author: huyn

"""

#304. Range Sum Query 2D - Immutable

#Given a 2D matrix matrix, find the sum of the elements inside the rectangle

#defined by its upper left corner (row1, col1) and lower right corner (row2, col2).

class NumMatrix:

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

self.arr = []

if matrix==[[[]]] or not matrix:

return

row = len(matrix)

col = len(matrix[0])

temp =[]

for c in range(col):

if temp:

temp.append(temp[-1]+matrix[0][c])

else:

temp.append(matrix[0][c])

self.arr.append(temp)

for r in range(1,row):

temp =[]

for c in range(1,col):

if not temp:

temp.append(matrix[r][c]+self.arr[r-1][c])

else:

temp.append(temp[c-1]+self.arr[r-1][c]-self.arr[r-1][c-1]+matrix[r][c])

self.arr.append(temp)

def sumRegion(self, row1: int, col1: int, row2: int, col2: int) -> int:

if col1 ==0:

if row1==0:

return self.arr[row2][col2]

else:

return self.arr[row2][col2]- self.arr[row1-1][col2]

else:

if row1==0:

return self.arr[row2][col2]-self.arr[row2][col1-1]

else:

return self.arr[row2][col2]- self.arr[row2][col1-1]-self.arr[row1-1][col2]+self.arr[row1-1][col1-1]

return# -\*- coding: utf-8 -\*-

"""

Created on Tue Sep 24 16:22:43 2019

@author: huyn

"""

#158. Read N Characters Given Read4 II - Call multiple times

#Given a file and assume that you can only read the file using a given method read4,

#implement a method read to read n characters. Your method read may be called multiple times.

#

#

#

#Method read4:

#

#The API read4 reads 4 consecutive characters from the file, then writes those characters into the buffer array buf.

#

#The return value is the number of actual characters read.

#

#Note that read4() has its own file pointer, much like FILE \*fp in C.

"""

The read4 API is already defined for you.

@param buf, a list of characters

@return an integer

def read4(buf):

# Below is an example of how the read4 API can be called.

file = File("abcdefghijk") # File is "abcdefghijk", initially file pointer (fp) points to 'a'

buf = [' '] \* 4 # Create buffer with enough space to store characters

read4(buf) # read4 returns 4. Now buf = ['a','b','c','d'], fp points to 'e'

read4(buf) # read4 returns 4. Now buf = ['e','f','g','h'], fp points to 'i'

read4(buf) # read4 returns 3. Now buf = ['i','j','k',...], fp points to end of file

"""

class Solution:

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

self.string = []

def read(self, buf, n):

"""

:type buf: Destination buffer (List[str])

:type n: Number of characters to read (int)

:rtype: The number of actual characters read (int)

"""

check = False

for i in range(min(n,len(self.string))):

buf[i]=self.string[i]

n-=1

check = True

if check:

i+=1

else:

i =0

self.string = self.string[i:]

while n>0:

b = [""]\*4

size = read4(b)

if size ==0:

return i

else:

for index in range(min(size,n)):

buf[i] = b[index]

i+=1

n-=1

for j in range(index+1,size):

self.string.append(b[j])

return i# -\*- coding: utf-8 -\*-

"""

Created on Thu Oct 10 11:07:11 2019

@author: Huy Nguyen

"""

#Read N Characters Given Read4

def read4(buf):

return

def read(buf, n):

"""

:type buf: Destination buffer (List[str])

:type n: Number of characters to read (int)

:rtype: The number of actual characters read (int)

"""

smallBuf = [""]\*4 # where we store the read4

i = 0

while n>0:

inputRead = read4(smallBuf)

for j in range(min(n,inputRead)):

buf[i]=smallBuf[j]

n-=1

i+=1

# we are done if either n==0 (which is check by the while loop condition) or inputRead less than 4 and we use all of them

if inputRead<4:

break

return i# -\*- coding: utf-8 -\*-

"""

Created on Thu Oct 10 11:36:03 2019

@author: Huy Nguyen

"""

#Read N Characters Given Read4 II

class Solution:

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

self.data =[] # this store extra info that read4 reads and our read did not need that much of data

def read(self, buf, n):

"""

:type buf: Destination buffer (List[str])

:type n: Number of characters to read (int)

:rtype: The number of actual characters read (int)

"""

smallBuf = [""]\*4

# firstly, since we might have some left overdata from previous read4 stored in data, we retrieve until either our self.data is empty or we fill n of buff

# create a flag to check whether we have use our data

usedData = False # this served as sometimes we only go through the loop once which makes i=0, but we actually need to make it 1

for i in range(min(n,len(self.data))):

buf[i] = self.data[i]

n-=1

usedData = True

if usedData:

i+=1

else:

i= 0

# we remove the amount that used from data

self.data = self.data[i:]

while n>0:

size = read4(smallBuf)

if size ==0: # dont have any more in input stream, we are done

break

for index in range(min(size,n)):

buf[i]=smallBuf[index]

i+=1

n-=1

# might be a case where our read4 is greater, means we read4 more than we need for read

# store it in our self.data

for j in range(index+1,size):

self.data.append(smallBuf[j])

return i# -\*- coding: utf-8 -\*-

"""

Created on Thu Oct 3 00:56:48 2019

@author: huyn

"""

#Given an array of positive and negative numbers, arrange them such that all negative

# integers appear before all the positive integers in the array without using any additional

# data structure like hash table, arrays, etc. The order of appearance should be maintained.

#

#Input: [12 11 -13 -5 6 -7 5 -3 -6]

#Output: [-13 -5 -7 -3 -6 12 11 6 5]

def rearrangeSlow(array):

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

if array[i]<0:

val = array[i]

j = i -1

while j>=0 and array[j]>0:

array[j+1]=array[j]

j-=1

array[j+1]= val

return

array = [12 ,11, -13, -5, 6, -7, 5, -3, -6]

#rearrangeSlow(array)

#print (array)

def rearrangeFast(array):

def dfs(start,stop):

if start<stop:

mid = (start+stop)//2

# print ("start:{},mid:{},stop:{}".format(start,mid,stop))

dfs(start,mid)

dfs(mid+1,stop)

# we find the first positive index and the end of it

leftPosStart = None

leftPosEnd = None

for i in range(start,stop+1):

if array[i]>0:

if leftPosStart==None:

leftPosStart = i

leftPosEnd = i

# we hit negative, ,we check if lefposStart still is None, keep continue

else:

if leftPosStart==None:

continue

else:

break

rightPosStart = None

rightPosEnd = None

if leftPosEnd!=None: # we find the first negative next to this

for i in range(min(leftPosEnd+1,stop),stop+1):

if array[i]<0:

if not rightPosStart:

rightPosStart = i

rightPosEnd=i

else:

break

if leftPosEnd!=None and rightPosStart!=None:

# we have LN LP RN RP

# reverse LP

reverseArr(array,leftPosStart,leftPosEnd)

# reverse RN

reverseArr(array,rightPosStart,rightPosEnd)

# reverse LP RN

reverseArr(array,leftPosStart,rightPosEnd)

dfs(0,len(array)-1)

return

def reverseArr(arr,start,stop):

while start<stop:

# print (start,stop)

arr[start],arr[stop]=arr[stop],arr[start]

start+=1

stop-=1

rearrangeFast(array)

# -\*- coding: utf-8 -\*-

"""

Created on Fri Sep 27 17:28:52 2019

@author: huyn

"""

#10. Regular Expression Matching

#

#Given an input string (s) and a pattern (p), implement regular expression matching with support for '.' and '\*'.

#

#'.' Matches any single character.

#'\*' Matches zero or more of the preceding element.

#The matching should cover the entire input string (not partial).

# T[i][j] =

# +T[i-1][j-1] if s[i]==p[j] or p[j]=="."

# +if pattern[j] == "\*" :

# - T[i][j-2] (not taking j-1 character in j, we care the boolean value 2 step back in pattern)

# - T[i][j-1] if s[i] == p[j-1] or pattern [j-1] =="."

# False

# Base case,

def isMatch(s: str, p: str) -> bool:

row = len(s)

col = len(p)

dp = [[False]\*(col+1) for i in range(row+1)] # accomodate the 0,0

# base case matching non string to non string

dp[0][0]= True

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

if p[i-1] == "\*":

dp[0][i] = dp[0][i-2] # xa\* depends on x match

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

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

if s[i-1] == p[j-1] or p[j-1]==".":

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

elif p[j-1]=="\*":

dp[i][j] = dp[i][j-2] # this assign True if the previous already True

if s[i-1] ==p[j-2] or p[j-2]==".":

dp[i][j] = dp[i-1][j] or dp[i][j] # if dp[i][j] was True already

else:

dp[i][j] = False

print (dp)

return dp[row][col]

s= "xaaby"

p = "xa\*b.c"

print (isMatch(s,p))

# a.b -> T acb, axb,atb

# a\*b -> T b, aab, aaab , F acb, a,

# a\*b.\*y -> T bzy,by,aabasdy, F ->

# 0 1 2 3 4 5 6

# x a \* b . c

# 0 T F F F F F F

# 1 x F T F T F F F

# 2 a F F T T F F F

# 3 a F F F T

# 4 b F

# 5 y F

# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 9 20:03:39 2019

@author: Huy Nguyen

"""

#Remove Duplicates from Sorted Array

#Given a sorted array nums, remove the duplicates in-place such that each element appear only once and return the new length.

#

#Do not allocate extra space for another array, you must do this by modifying the input array in-place with O(1) extra memory.

def removeDuplicates(nums):

"""

:type nums: List[int]

:rtype: int

"""

if len(nums)<=1:

return len(nums)

count=1 # count how many differences number

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

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

count+=1

# do something to swap the good up

# the number of count also indicate where the new number is (index i), and where we should put this number at (at index count-1)

nums[count-1]= nums[i]

return count# -\*- coding: utf-8 -\*-

"""

Created on Sun Sep 22 17:14:59 2019

@author: huyn

"""

#301. Remove Invalid Parentheses

#Remove the minimum number of invalid parentheses in order to make the input string valid. Return all possible results.

#

#Note: The input string may contain letters other than the parentheses ( and ).

def removeInvalidParentheses(s: str):

# find misplace left and right

right = 0

left = 0

correct =0

for item in s:

if item=="(":

left+=1

elif item==")":

if left>0:

left-=1

correct+=1

elif left==0:

right+=1

res = set()

def dfs(left,right,currentList,index,correctLeft,correctRight):

# when index is equal to length of s, we have 1 solution

if index==len(s):

# print (30,left,right,currentList,index,correctLeft,correctRight)

res.add("".join(currentList))

elif index<len(s):

currentChar = s[index]

# print (37,currentChar,left,right,currentList,index,correctLeft,correctRight)

if currentChar == "(":

# we check if we can ignore this

if left>0:

# this means we can choose to skip this

dfs(left-1,right,currentList,index+1,correctLeft,correctRight)

# we will include it if and only if correctLeft greater than 0

if correctLeft>0:

currentList.append("(")

dfs(left,right,currentList,index+1,correctLeft-1,correctRight)

currentList.pop()

elif currentChar==")":

if right>0:

# this means we can choose to skip this

dfs(left,right-1,currentList,index+1,correctLeft,correctRight)

if correctRight>0 and correctLeft<correctRight: # make sure what we have is still valid

currentList.append(")")

dfs(left,right,currentList,index+1,correctLeft,correctRight-1)

currentList.pop()

else:

currentList.append(currentChar)

dfs(left,right,currentList,index+1,correctLeft,correctRight)

currentList.pop()

dfs(left,right,[],0,correct,correct)

return list(res)

s= "(a)())()"

print (removeInvalidParentheses(s))

s ="(a)())()"# -\*- coding: utf-8 -\*-

"""

Created on Thu Oct 10 16:08:25 2019

@author: Huy Nguyen

"""

#Given a singly linked list L: L0→L1→…→Ln-1→Ln,

#reorder it to: L0→Ln→L1→Ln-1→L2→Ln-2→…

#

#You may not modify the values in the list's nodes, only nodes itself may be changed.

def reorderList( head) -> None:

"""

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

"""

# count length

n = 0

cur = head

while cur:

n += 1

cur = cur.next

# if just have lest or equal to 2, everything stays the same

if n < 3:

return head

# find head of and reverse second half

first = second = head

for \_ in range((n + 1) // 2):

second = second.next

# reverse the second half into somehting like n->n-1>n-2...->n/2

second = reverse(second)

# merge

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

# get the nexf both to link

first\_next, second\_next = first.next, second.next

# 1->2 ->3 ->4

# 1->2 , 4->3

# link 4 to 2

second.next = first.next

# link 1 to 4

first.next = second

# we reset our first,second to be the head of 2,3

first = first\_next

second = second\_next

# deal with last pair and possible last node

last = None if n % 2 == 0 else first.next # we dont care about last if we have even node, else, it should be the first.next

# 1->2->3

# 5-> 4->3

first.next = second # link 2->4

if last:

# we have to set second next (4->3), and set last next to None 1->5->2->4->3

second.next = last

last.next = None

def reverse(head) :

prev, cur = None, head

while cur:

next\_ = cur.next

cur.next = prev

prev = cur

cur = next\_

return prev# -\*- coding: utf-8 -\*-

"""

Created on Thu Oct 10 15:01:07 2019

@author: Huy Nguyen

"""

def reverseList(head):

def dfs(head,prev):

if not head:

return head,head

else:

newHead, newRoot = dfs(head.next,head) # our recursive relation

# our work to link

# link our head.next to prev

head.next = prev

# link newHead next to head

newHead.next = head

return head,newRoot

if head:

return dfs(head,None)[1]

else:

return None# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 9 16:49:16 2019

@author: huyn

"""

def romanToInt(s: str) -> int:

d= {"I":1,"V":5,"X":10,"L":50,"C":100,"D":500,"M":1000}

if not s:

return 0

current = d[s[0]]

accumulate = d[s[0]]

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

v = d[s[i+1]]

if v>current:

accumulate= accumulate-current+v-current

else:

accumulate+=v

current=v

return accumulate# -\*- coding: utf-8 -\*-

"""

Created on Fri Oct 11 01:59:50 2019

@author: huyn

"""

#Search in Rotated Sorted Array

def search(nums,target):

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

while start<stop:

return# -\*- coding: utf-8 -\*-

"""

Created on Thu Sep 19 14:48:53 2019

@author: huyn

"""

#34. Find First and Last Position of Element in Sorted Array

def searchRange(nums,target):

start,stop = findLeftMost(nums,target),findRightMost(nums,target)

return [start,stop]

def findLeftMost(nums,target):

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

while start+1<stop:

mid = (start+stop)//2

if nums[mid]>=target:

stop=mid

else:

start = mid

if nums[start]==target:

return start

if nums[stop]==target:

return stop

return -1

def findRightMost(nums,target):

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

while start+1<stop:

mid = (start+stop)//2

if nums[mid]<=target:

start=mid

else:

stop = mid

if nums[stop]==target:

return stop

if nums[start]==target:

return start

return -1# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 9 11:17:30 2019

@author: huyn

"""

class segmentNode:

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

self.start = start

self.stop = stop

self.val = val

self.left = left

self.right = right

class segmentTree:

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

self.root = self.initialize(arr,0,len(arr)-1)

def initialize(self,nums,start,stop):

if start == stop:

node = segmentNode(nums[start],start,stop)

return node

elif start<stop:

mid = (start+stop)//2

left = self.initialize(nums,start,mid)

right = self.initialize(nums,mid+1,stop)

node = segmentNode(left.val+right.val,start,stop,left,right)

return node

def getSum(self,i,j):

root = self.root

def dfs(root,i,j):

start,stop = root.start,root.stop

if i== start and j == stop:

return root.val

else:

mid = (start+stop)//2

val = 0

# check if i>mid

if i>mid:

val+=dfs(root.right,i,j)

elif i<=mid:

if j<=mid:

val+=dfs(root.left,i,j)

else:

# this means we have to search bothway

val+=dfs(root.left,i,mid)

val+=dfs(root.right,mid+1,j)

return val

return dfs(root,i,j)

def update(self,index,val):

root = self.root

def dfs(root,index,val):

if root.start == root.stop ==index:

extra = val-root.val

root.val = val

return extra

else:

mid = (root.start+root.stop)//2

if mid<index:

extra = dfs(root.right,index,val)

else:

extra = dfs(root.left,index,val)

root.val+=extra

return extra

extra = dfs(root,index,val)

return extra

arr = [1,3,5,7,9,11,13]

tree = segmentTree(arr)

#print (tree.root.val)

print (tree.getSum(0,4))

tree.update(3,100)

print (tree.getSum(3,3))# -\*- coding: utf-8 -\*-

"""

Created on Sat Sep 21 14:27:37 2019

@author: huyn

"""

from collections import deque

# Definition for a binary tree node.

class TreeNode(object):

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

self.val = x

self.left = None

self.right = None

class CodecBFS:

def serialize(self, root):

"""Encodes a tree to a single string.

:type root: TreeNode

:rtype: str

"""

res= []

queue = deque([root])

while queue:

node = queue.popleft()

if not node:

res.append("n")

continue

else:

res.append(node.val)

if node.left:

queue.append(node.left)

else:

queue.append(None)

if node.right:

queue.append(node.right)

else:

queue.append(None)

return ",".join([str(i) for i in res])

def deserialize(self, data):

"""Decodes your encoded data to tree.

:type data: str

:rtype: TreeNode

"""

data= deque(data.split(","))

if len(data)==1:

if data[0]=="n":

return None

root = TreeNode(int(data.popleft()))

head = root

currentLevel = deque([head])

while currentLevel:

size = len(currentLevel)

nextLevel = []

# print (59,currentLevel,data,size)

for i in range(size):

node = currentLevel[i]

left = data.popleft()

right = data.popleft()

if left !="n":

leftNode = TreeNode(int(left))

node.left = leftNode

nextLevel.append(leftNode)

if right!="n":

rightNode = TreeNode(int(right))

node.right = rightNode

nextLevel.append(rightNode)

currentLevel = nextLevel

return root

#a= TreeNode(1)

#b=TreeNode(2)

#c= TreeNode(3)

#d= TreeNode(4)

#e = TreeNode(5)

#f= TreeNode(6)

#g = TreeNode(7)

#a.left = b

#a.right = c

#b.left = d

#b.right = e

#e.left = f

#e.right= g

#serialize = CodecBFS()

#data=serialize.serialize(a)

#root = serialize.deserialize(data)

class CodecPreorder:

def serialize(self, root):

"""Encodes a tree to a single string.

:type root: TreeNode

:rtype: str

"""

myList =[]

def dfs(root):

if root:

myList.append(str(root.val))

dfs(root.left)

dfs(root.right)

else:

myList.append("#")

dfs(root)

return ",".join(myList)

def deserialize(self, data):

"""Decodes your encoded data to tree.

:type data: str

:rtype: TreeNode

"""

data= data.split(",")

# print (data)

def dfs(data,index):

if data[index]=="#":

return None,1

else:

val = int(data[index])

newNode = TreeNode(val)

left,countL = dfs(data,index+1)

right,countR = dfs(data,index+countL+1)

newNode.left = left

newNode.right = right

return newNode,countL+countR+1

return dfs(data,0)[0]

def deserializeIterator(self, data):

"""Decodes your encoded data to tree.

:type data: str

:rtype: TreeNode

"""

data= data.split(",")

# print (data)

def dfs(iterator):

val=next(iterator)

if val=="#":

return None

else:

val = int(val)

newNode = TreeNode(val)

newNode.left= dfs(iterator)

newNode.right= dfs(iterator)

return newNode

return dfs(iter(data))

#a= TreeNode(1)

#b=TreeNode(2)

#c= TreeNode(3)

#d= TreeNode(4)

#e = TreeNode(5)

#f= TreeNode(6)

#g = TreeNode(7)

#a.left = b

#a.right = c

#b.left = d

#b.right = e

#e.left = f

#e.right= g

#serialize = CodecPreorder()

#data=serialize.serialize(a)

#root = serialize.deserializeIterator(data)

#Approach 1C: DFS preorder with non-NULL number of children info

class CodecNotNull:

def serialize(self, root):

"""Encodes a tree to a single string.

:type root: TreeNode

:rtype: str

"""

myList =[]

def dfs(root):

if root:

myList.append(str(root.val))

num = 0

if root.left:

num+=1

if root.right:

num+=1

myList.append(str(num))

dfs(root.left)

dfs(root.right)

dfs(root)

return ",".join(myList)

def deserialize(self, data):

"""Decodes your encoded data to tree.

:type data: str

:rtype: TreeNode

"""

data= data.split(",")

# print (data)

def dfs(data,index):

nodeVal = int(data[index])

numChild = int(data[index+1])

newNode = TreeNode(nodeVal)

leftNum,rightNum =0,0

# print (index,leftNum,rightNum)

if numChild>=1:

newNode.left,leftNum = dfs(data,index+2)

if numChild==2:

newNode.right,rightNum = dfs(data,index+2+leftNum)

# print (index,leftNum,rightNum)

return newNode,leftNum+rightNum

return dfs(data,0)[0]

def deserializeIterator(self, data):

"""Decodes your encoded data to tree.

:type data: str

:rtype: TreeNode

"""

data= data.split(",")

# print (data)

def dfs(iterator):

try:

nodeVal = int(next(iterator))

numChild = int(next(iterator))

newNode = TreeNode(nodeVal)

if numChild>=1:

newNode.left = dfs(iterator)

if numChild==2:

newNode.right = dfs(iterator)

return newNode

except:

return None

return dfs(iter(data))

a= TreeNode(1)

b=TreeNode(2)

c= TreeNode(3)

d= TreeNode(4)

e = TreeNode(5)

f= TreeNode(6)

g = TreeNode(7)

a.left = b

a.right = c

b.left = d

b.right = e

e.left = f

e.right= g

serialize = CodecNotNull()

data=serialize.serialize(a)

root = serialize.deserialize(data)

#print (serialize.serialize(root) ==data)# -\*- coding: utf-8 -\*-

"""

Created on Sun Sep 22 23:06:37 2019

@author: huyn

"""

#317. Shortest Distance from All Buildings

#You want to build a house on an empty land which reaches all buildings in the shortest amount of distance.

#You can only move up, down, left and right. You are given a 2D grid of values 0, 1 or 2, where:

#

#Each 0 marks an empty land which you can pass by freely.

#Each 1 marks a building which you cannot pass through.

#Each 2 marks an obstacle which you cannot pass through.

def shortestDistance(grid):

return

grid = [[1,0,2,0,1],[0,0,0,0,0],[0,0,1,0,0]]

# -\*- coding: utf-8 -\*-

"""

Created on Sun Oct 6 19:27:25 2019

@author: huyn

"""

#71. Simplify Path

#Given an absolute path for a file (Unix-style), simplify it. Or in other words,

# convert it to the canonical path.

#

#In a UNIX-style file system, a period . refers to the current directory. Furthermore,

#a double period .. moves the directory up a level. For more information, see:

# Absolute path vs relative path in Linux/Unix

#

#Note that the returned canonical path must always begin with a slash /, and there

#must be only a single slash / between two directory names. The last directory name

#(if it exists) must not end with a trailing /. Also, the canonical path must be the shortest

#string representing the absolute path.

def simplifyPath(self, path: str) -> str:

path = [item for item in path.split("/") if item!=""]

stack = []

for item in path:

if item ==".":

continue

elif item == "..":

if stack:

stack.pop()

else:

stack.append(item)

if not stack:

return "/"

return "/"+"/".join(stack)

# -\*- coding: utf-8 -\*-

"""

Created on Sun Sep 29 20:40:26 2019

@author: Huy Nguyen

"""

#Sorted Iterator

#ou have three unsorted list of numbers. Design/Write function that will return next minimum element out of these lists (remove that element from list).

list1 = [5, 1, 2, 4]

list2 = [4, 6, 3]

list3 = [9, 0, 7]

import heapq

class Iterator:

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

self.lists = lists

self.heap = []

for index,eachList in enumerate(self.lists):

eachList.sort(reverse= True)

heapq.heappush(self.heap,(eachList.pop(),index))

def next(self):

if self.heap:

val,index= heapq.heappop(self.heap)

if self.lists[index]:

heapq.heappush(self.heap,(self.lists[index].pop(),index))

return val

iterator = Iterator([list1,list2,list3])

print (iterator.next())

print (iterator.next())

print (iterator.next())

print (iterator.next())

print (iterator.next())

print (iterator.next())

print (iterator.next())

print (iterator.next())

print (iterator.next())

print (iterator.next())# -\*- coding: utf-8 -\*-

"""

Created on Sun Sep 29 21:00:27 2019

@author: Huy Nguyen

"""

#Spaced permutations

#Given an integer n, create an array such that each value is repeated twice.

def generateRepeate(n):

arr =[]

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

for j in range(2):

arr.append(i)

return arr

#Follow up 1: After creating it, find a permutation such that each number is spaced in

# such a way, they are at a "their value" distance from the second occurrence of the same number.

# Return any 1 permutation if it exists. Empty array if no permutation exists.

#Input: n = 3 --> This is the array - [1, 1, 2, 2, 3, 3]

#Output: [3, 1, 2, 1, 3, 2]

#Explanation:

#The second 3 is 3 digits away from the first 3.

#The second 2 is 2 digits away from the first 2.

#The second 1 is 1 digit away from the first 1

def findPermutation(n):

arr = [0]\*(2\*n)

def dfs(num):

if num ==0:

return True

for i in range(2\*n-num-1):

try:

if arr[i]==0 and arr[i+num+1]==0:

arr[i],arr[i+num+1] = num,num

check = dfs(num-1)

if check:

return True

arr[i],arr[i+num+1] = 0,0

except:

continue

return False

if dfs(n):

return arr

return False

#print (findPermutation(1))

#

#print (findPermutation(2))

#for i in range(1,11):

# print (findPermutation(i))

#Follow up 2: Return all possible permutations.

def findAllPermute(n):

res = []

arr = [0]\*(2\*n)

def dfs(num,arr):

if num ==0:

t= []

for num in arr:

t.append(num)

res.append(t)

for i in range(2\*n-num-1):

try:

if arr[i]==0 and arr[i+num+1]==0:

arr[i],arr[i+num+1] = num,num

dfs(num-1,arr)

arr[i],arr[i+num+1] = 0,0

except:

continue

dfs(n,arr)

if res:

return res

return False

#print (findAllPermute(8))# -\*- coding: utf-8 -\*-

"""

Created on Thu Sep 19 12:43:31 2019

@author: huyn

"""

# given number n, print spiral matrix n

def printSpiral(n):

output = [[0]\*n for i in range(n)]

directions=[[0,1],[1,0],[0,-1],[-1,0]]

current = (0,0)

index = 0

def check(x,y,direction,n):

addX,addY = direction

X,Y =x+addX,y+addY

if X>=0 and Y>=0 and X<n and Y<n and output[X][Y]==0:

return True

return False

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

x,y = current

output[x][y]=i

# plan for next step

direction = directions[index]

if check(x,y,direction,n):

current = (x+direction[0],y+direction[1])

else:

index = (index+1)%4

direction = directions[index]

current = (x+direction[0],y+direction[1])

return output

#print (printSpiral(3))# -\*- coding: utf-8 -\*-

"""

Created on Fri Oct 11 01:40:37 2019

@author: huyn

"""

#Strobogrammatic Number II

def findStrobogrammatic(n: int) :

mapping = {"1":"1","8":"8","9":"6","6":"9","0":"0"}

if n ==0 :

return [""]

elif n==1:

return ["1","8","0"]

res = set()

def dfs(path,index):

if index == n//2:

string = "".join(path)

reverseString = ""

for item in string[::-1]:

reverseString+=mapping[item]

if n%2:

string1 = string+"1"+reverseString

res.add(string1)

string2 = string+"8"+reverseString

res.add(string2)

string3 = string+"0"+reverseString

res.add(string3)

else:

string1 = string+reverseString

res.add(string1)

string2 = string+reverseString

res.add(string2)

elif index<n//2:

if index ==0:

for item in mapping:

if item!="0":

path.append(item)

dfs(path,index+1)

path.pop()

else:

for item in mapping:

path.append(item)

dfs(path,index+1)

path.pop()

dfs([],0)

return [item for item in res] # -\*- coding: utf-8 -\*-

"""

Created on Thu Sep 19 01:05:34 2019

@author: huyn

"""

import random

#560. Subarray Sum Equals K

#Given an array of integers and an integer k, you need to find the total number

# of continuous subarrays whose sum equals to k.

def subarraySum( nums, k) -> int:

# solve by checking accumulate

# initialize of dictionary that store how many time before a certain continous array from 0 to i,j,k,... sums up to value x

dictionary = {}

# store 0 as 1, for case like k= 1 , nums[0] = 1

dictionary[0]=1

currentSum = 0

count = 0

for num in nums:

currentSum +=num

val = currentSum -k

if val in dictionary:

count+=dictionary[val] # we add to the count how may time it can form a continous subs array

if currentSum not in dictionary:

dictionary[currentSum]= 0

dictionary[currentSum]+=1 # we increase number of contnous array have same sum

return count

def generateTest(n):

arr = []

for i in range(n):

arr.append(random.randint(-10,10))

return arr

arr,k = generateTest(100),random.randint(1,100)

print (arr,k)

# -\*- coding: utf-8 -\*-

"""

Created on Thu Sep 19 23:38:09 2019

@author: Huy Nguyen

"""

import random

#523. Continuous Subarray Sum

#Given a list of non-negative numbers and a target integer k, write a function to

#check if the array has a continuous subarray of size at least 2 that sums up to a multiple of k, that is, sums up to n\*k where n is also an integer.

def checkSubarraySum(nums,k):

d = {}

s = 0

for i in range(len(nums)):

if i==0:

try:

s=nums[i]%k

except:

s= nums[i]

d[s]=[0]

else:

try:

s= (s+nums[i])%k

except:

s+=nums[i]

if s==0:

return True

if s not in d:

d[s]=[]

# check if index at least 2

for index in d[s]:

if abs(index-i)>=2:

return True

d[s].append(i)

return False

def generate(n):

arr = []

for i in range(n):

arr.append(random.randint(-10,30))

return arr

nums,k = generate(100),100000

print (nums)

print (k)

# -\*- coding: utf-8 -\*-

"""

Created on Fri Oct 11 01:38:35 2019

@author: huyn

"""

#Subsets

def subsets(nums):

if not nums:

return []

res = [[]]

for num in nums:

temp = []

for item in res:

temp.append(item[:])

item.append(num)

temp.append(item[:])

res = temp

return res

print (subsets([1,2,3]))# -\*- coding: utf-8 -\*-

"""

Created on Fri Oct 11 12:45:59 2019

@author: huyn

"""

def subsetSumDP(arr,K):

row = len(arr)+1

col = K+1

dp = [[False]\*col for i in range(row)]

for r in range(row):

dp[r][0]=True

for r in range(1,row):

for c in range(1,col):

dp[r][c] = dp[r-1][c] or dp[r-1][c-arr[r-1]]

return dp[row-1][col-1]

arr = [4,5,0,2,3,1]

K = 5

#print (subsetSumDP(arr,K))

def subsetSumBacktrack(arr,K):

arr.sort()

def dfs(accumulate,index,currentPath,K):

if accumulate == K:

print (currentPath)

elif accumulate<K and index<len(arr):

val = accumulate+arr[index]

if val<=K:

currentPath.append(arr[index])

dfs(val,index+1,currentPath,K)

currentPath.pop()

dfs(accumulate,index+1,currentPath,K)

dfs(arr[0],1,[arr[0]],K)

dfs(0,1,[],K)

return

#subsetSumBacktrack(a# -\*- coding: utf-8 -\*-

"""

Created on Thu Oct 10 15:57:50 2019

@author: Huy Nguyen

"""

def swapPairs(head) :

if head and head.next:

node = swapPairs(head.next.next)

temp = head.next

temp.next = head

head.next = node

head = temp

return head

elif head:

return head

else:

return None

# -\*- coding: utf-8 -\*-

"""

Created on Sun Oct 6 19:28:53 2019

@author: huyn

"""

#621. Task Scheduler

#Given a char array representing tasks CPU need to do. It contains capital letters

#A to Z where different letters represent different tasks. Tasks could be done without

#original order. Each task could be done in one interval. For each interval, CPU could

#finish one task or just be idle.

#

#However, there is a non-negative cooling interval n that means between two same tasks,

#there must be at least n intervals that CPU are doing different tasks or just be idle.

#

#You need to return the least number of intervals the CPU will take to finish all the given tasks.

import heapq

from typing import List

def leastInterval(tasks: List[str], n: int) -> int:

d = {}

for task in tasks:

if task not in d:

d[task]=0

d[task]+=1

priorityQueue= []

time = 0

for task in d:

heapq.heappush(priorityQueue,-d[task])

print (priorityQueue)

while priorityQueue:

jobs = []

for i in range(n+1):

if priorityQueue:

jobs.append(heapq.heappop(priorityQueue))

print (jobs)

for job in jobs:

job=job+1

if job!=0:

heapq.heappush(priorityQueue,job)

if not priorityQueue:

time+=len(jobs)

else:

time+=n+1

return time

#tasks = ["A","A","A","B","B","B"]

#n = 2

#print (leastInterval(tasks,n))# -\*- coding: utf-8 -\*-

"""

Created on Fri Oct 4 03:27:24 2019

@author: huyn

"""

class Node:

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

self.val = val

self.left = left

self.right = right

self.parent = parent

def print(self):

root = self

def dfs(root):

if root:

print (root.val)

dfs(root.left)

dfs(root.right)

dfs(root)

# Toggle Node

def toggle(node):

# if node is 0, then basically made everything on the path to 0

if node.val ==1:

while node:

node.val = 0

node = node.parent

else:

# togle to 1

parent = node.parent

node.val = 1

while parent:

if parent.left.val==1 and parent.right.val==1:

parent.val =1

else:

parent.val =0

parent = parent.parent

#

#A = Node(1)

#B= Node(1)

#C=Node(1)

#D=Node(1)

#E=Node(1)

#F=Node(1)

#G=Node(1)

#A.left = B

#A.right = C

#B.parent=A

#C.parent=A

#B.left = D

#B.right = E

#D.parent = B

#E.parent=B

#C.left =F

#C.right = G

#F.parent=C

#G.parent= F

#A.print()

#toggle(E)

#A.print()

# -\*- coding: utf-8 -\*-

"""

Created on Tue Sep 24 14:00:52 2019

@author: huyn

"""

from collections import deque

#topology sorting

class Node:

def \_\_init\_\_(self,val,neighbor=[]):

self.val = val

self.neighbors = neighbor

def topologySort(nodes):

res = []

# go through the dictionary, find the parent for all the node

parentNodeOf = {}

for node in nodes:

if node not in parentNodeOf:

parentNodeOf[node] = set()

for neighbor in node.neighbors:

if neighbor not in parentNodeOf:

parentNodeOf[neighbor] = set()

parentNodeOf[neighbor].add(node)

# list of node without incoming edges

# print ("parentNodeOf",parentNodeOf)

startNode =[node for node in parentNodeOf if len(parentNodeOf[node])==0]

# print (startNode)

while startNode:

node = startNode.pop()

res.append(node)

for neighbor in node.neighbors:

# remove the parent node from this neighbor

parentNodeOf[neighbor].remove(node)

if not parentNodeOf[neighbor]: # if no more, then this becamses a start node

startNode.append(neighbor)

return [item.val for item in res]

a= Node(5)

b= Node(7)

c= Node(3)

d= Node(11)

e= Node(8)

f= Node(2)

g= Node(9)

h= Node(10)

a.neighbors= [d]

b.neighbors=[d,e]

c.neighbors=[e,h]

d.neighbors=[f,g,h]

e.neighbors=[g]

nodes = [a,b,c,d,e,f,g,h]

res = topologySort(nodes)

# -\*- coding: utf-8 -\*-

"""

Created on Fri Sep 20 11:47:21 2019

@author: huyn

"""

#42. Trapping Rain Water

#Given n non-negative integers representing an elevation map where the width of

#each bar is 1, compute how much water it is able to trap after raining.

# naive way, for each coloum, we count how much that column can store water by looking for max left

# and max right

def trapNaive(height) -> int:

rain = 0

for index,water in enumerate(height):

maxL = max(height[:index+1])

maxR = max(height[index:])

rain+= min(maxL,maxR)-water

return rain

# using same idea, but store the maxL,maxR at each index

def trapDynamic(height):

left = []

for index,water in enumerate(height):

if not left:

left.append(water)

else:

left.append(max(left[-1],water))

right = []

for index in range(len(height)-1,-1,-1):

water = height[index]

if not right:

right.append(water)

else:

right.append(max(right[-1],water))

size = len(height)

water= 0

for i in range(size):

water+=min(left[i],right[size-1-i])-height[i]

return water

# using same idea, but with pointers

def trapPointers(height):

water= 0

return water

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

#print (trapNaive(height))

#print (trapDynamic(height))# -\*- coding: utf-8 -\*-

"""

Created on Mon Sep 16 22:17:19 2019

@author: huyn

"""

class Node:

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

self.val = val

self.left = left

self.right = right

A= Node(4,None,None)

B= Node(5,None,None)

C= Node(2,None,None)

D= Node(1,None,None)

E= Node(3,None,None)

A.left = C

A.right = B

C.left = D

C.right= E

#426.

#Convert a BST to a sorted circular doubly-linked list in-place. Think of the left

#and right pointers as synonymous to the previous and next pointers in a doubly-linked list.\

minNode = None

maxNode = None

def treeToDoublyList(root: 'Node') -> 'Node':

if not root:

return None

def dfs(root):

nonlocal myMin,myMax

if root:

dfs(root.left)

# logic for root

if not myMax: # means that we hit the first root with val value, and we did not store our myMax as the last node yet, therefore, it is minimum

myMin = root

else:

# already have the last max, we can point it to our node, and node.left to max

root.left =myMax

myMax.right = root

# set our root to our max now

myMax = root

dfs(root.right)

myMin,myMax = None,None

dfs(root)

myMin.left = myMax

myMax.right= myMin

return root

treeToDoublyList(A)

# -\*- coding: utf-8 -\*-

"""

Created on Tue Oct 8 21:04:21 2019

@author: Huy Nguyen

"""

from collections import deque

class TreeNode:

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

self.val =val

self.left = None

self.right = None

n1 = TreeNode(1);

n2 = TreeNode(2);

n3 = TreeNode(3);

n4 = TreeNode(4);

n5 = TreeNode(5)

n6 = TreeNode(6)

n1.left = n2

n1.right = n3

n3.left = n4

arr= [n1,n2,n3]

def isBinaryTree(arr):

parents= {}

# check for if a node has more than 1 parent

for node in arr:

if node.left:

if node.left not in parents:

parents[node.left] = node

else:

return False

if node.right:

if node.right not in parents:

parents[node.right] = node

else:

return False

# check if there are more than 1 tree by traversing each node to the root node

# if there are more than 1 root node, then there is a problem

nodeSet=set(arr)

rootNodes = set()

while nodeSet:

randomNode = nodeSet.pop()

# traverse this randomNode up until we can't do it anymore

while randomNode in parents:

randomNode = parents[randomNode]

if randomNode in nodeSet:

nodeSet.remove(randomNode)

rootNodes.add(randomNode)

if len(rootNodes)!=1:

return False

# get the rootNode

root = rootNodes.pop()

# traverse through root, check if there is a cycle

visited= set()

visited.add(root)

queue = deque([root])

while queue:

size = len(queue)

for i in range(size):

node = queue.popleft()

# print (node)

if node.left:

if node.left in visited:

return False

visited.add(node.left)

queue.append(node.left)

if node.right:

if node.right in visited:

return False

visited.add(node.right)

queue.append(node.right)

return len(visited)==len(arr)

print (isBinaryTree(arr))# -\*- coding: utf-8 -\*-

"""

Created on Thu Oct 10 13:50:19 2019

@author: Huy Nguyen

"""

# valid ip address

def validIPAddress (IP: str) -> str:

if isIP4(IP):

return "IPv4"

if isIP6(IP):

return "IPv6"

return "Neither"

def isIP4(string):

string = string.split(".")

if len(string)!=4:

return False

for item in string:

if len(item)==1:

continue

else:

check = True

if item.isdigit():

check= (item[0]!="0" and int(item)<256)

else:

return False

if not check:

return False

return True

def isIP6(string):

string = string.split(":")

if len(string)!=8:

return False

for item in string:

if item==0:

if len(item)>1:

if item[0]=="0":

return False

if len(item)>4:

return False

if len(item)==1:

if item.isalpha():

if item.lower()>"f":

return False

elif len(item)>1:

for l in item:

if l.isalpha():

if l.lower()>"f":

return False

elif l.isdigit():

continue

else:

return False

else:

return False

return True

#IP = "172.16.254.1" ->True

#IP = "g:f:f:f:f:f:f:g" ->False

#IP = "02001:0db8:85a3:0000:0000:8a2e:0370:7334" ->False

#IP = "2001:db8:85a3:0:0:8A2E:0370:7334" ->True

# -\*- coding: utf-8 -\*-

"""

Created on Wed Oct 9 01:30:16 2019

@author: huyn

"""

#680. Valid Palindrome II

#Given a non-empty string s, you may delete at most one character. Judge whether you can make it a palindrome.

def validPalindrome(s: str) -> bool:

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

def dfs(s,start,stop,count):

if start == stop:

return True

elif start+1==stop:

if s[start]==s[stop]:

return True

else:

return count == 0

elif start<stop:

if s[start]==s[stop]:

return dfs(s,start+1,stop-1,count)

else:

if count>0:

return False

else:

return dfs(s,start+1,stop,1) or dfs(s,start,stop-1,1)

return dfs(s,start,stop,0)

def validPalindromeI(s: str) -> bool:

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

count = 0

return dfs(s,start,stop,0)# -\*- coding: utf-8 -\*-

"""

Created on Thu Sep 19 11:05:57 2019

@author: huyn

"""

from collections import deque

from typing import List

#314. Binary Tree Vertical Order Traversal

#Given a binary tree, return the vertical order traversal of its nodes' values.

#(ie, from top to bottom, column by column).

#

#If two nodes are in the same row and column, the order should be from left to right.

class TreeNode(object):

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

self.val = x

self.left = left

self.right = right

def verticalOrder(root):

arr = {}

queue = deque([(root,0)])

while queue:

node,level = queue.popleft()

if node.left:

queue.append([node.left,level-1])

if node.right:

queue.append([node.right,level+1])

if level not in arr:

arr[level]=[]

arr[level].append(node.val)

return [arr[i] for i in sorted(arr)]

def verticalTraversal(root: TreeNode) -> List[List[int]]:

if not root:

return []

arr = {}

queue = deque([(root,0,0)])

while queue:

node,level,horizontal = queue.popleft()

if node.left:

queue.append([node.left,level-1,horizontal+1])

if node.right:

queue.append([node.right,level+1,horizontal+1])

if level not in arr:

arr[level]=[]

arr[level].append((node.val,horizontal))

res= []

for key in sorted(arr):

myList = arr[key]

temp = []

for item in sorted(myList,key=lambda x:(x[1],x[0])):

temp.append(item[0])

res.append(temp)

return res# -\*- coding: utf-8 -\*-

"""

Created on Tue Oct 8 14:24:33 2019

@author: huyn

"""

#286. Walls and Gates

#Given a maze with cells being: gates, walls or empty spaces.

#You are given a m x n 2D grid initialized with these three possible values.

#-1 - A wall or an obstacle.

#0 - A gate.

#INF - Infinity means an empty room. We use the value 231 - 1 = 2147483647 to represent INF as you

#may assume that the distance to a gate is less than 2147483647.

#Fill each empty room with the distance to its nearest gate. If it is impossible to reach a gate,

#it should be filled with INF.

from typing import List

from collections import deque

num = 2147483647

rooms=[[2147483647,-1,0,2147483647],[2147483647,2147483647,2147483647,-1],[2147483647,-1,2147483647,-1],[0,-1,2147483647,2147483647]]

def wallsAndGates(rooms: List[List[int]]) -> None:

rows = len(rooms)

cols = len(rooms[0])

for r in range(rows):

for c in range(cols):

# find the gate

if rooms[r][c]==0:

visited = set()

distance = 0

queue = deque([[r,c]])

# print (queue)

visited.add((r,c))

while queue:

size = len(queue)

distance +=1

for i in range(size):

currentRow,currentCol = queue.popleft()

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

for x,y in add:

nextRow,nextCol = currentRow+x,currentCol+y

if isValid(nextRow,nextCol,rows,cols) and (nextRow,nextCol) not in visited:

if rooms[nextRow][nextCol]!=-1 and rooms[nextRow][nextCol]!=0:

visited.add((nextRow,nextCol))

queue.append((nextRow,nextCol))

# update rooms distance

rooms[nextRow][nextCol] = min(rooms[nextRow][nextCol],distance)

return

def isValid(currentRow,currentCol,rows,cols):

return currentRow>=0 and currentCol>=0 and currentRow<rows and currentCol<cols

#wallsAndGates(rooms)

def wallsAndGatesCheckAllSameTime(rooms: List[List[int]]) -> None:

rows = len(rooms)

cols = len(rooms[0])

gates = []

for r in range(rows):

for c in range(cols):

# find the gate

if rooms[r][c]==0:

gates.append([[r,c]])

visited = set()

distance = 0

while gates:

# print (gates)

temp =[]

distance+=1

for item in gates:

nextList = []

for currentRow,currentCol in item:

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

for x,y in add:

nextRow,nextCol = currentRow+x,currentCol+y

if isValid(nextRow,nextCol,rows,cols) and (nextRow,nextCol) not in visited:

if rooms[nextRow][nextCol]!=-1 and rooms[nextRow][nextCol]!=0:

visited.add((nextRow,nextCol))

nextList.append((nextRow,nextCol))

rooms[nextRow][nextCol] =distance

if nextList:

temp.append(nextList)

gates= temp

return

wallsAndGatesCheckAllSameTime(rooms)# -\*- coding: utf-8 -\*-

"""

Created on Fri Oct 4 03:39:55 2019

@author: huyn

"""

#291. Word Pattern II

#Given a pattern and a string str, find if str follows the same pattern.

#

#Here follow means a full match, such that there is a bijection between a letter

# in pattern and a non-empty substring in str.

def wordPatternMatchNaive(pattern: str, string: str) -> bool:

def dfs(pToS,StoP,indexPattern,indexString):

if indexPattern==len(pattern) and indexString==len(string):

# print (pToS,StoP)

return True

elif indexPattern<len(pattern) and indexString<len(string):

# we can keep traverse

p = pattern[indexPattern]

if p in pToS:

#check if it maps out for the string

for letter in pToS[p]:

if indexString == len(string):

# print (26,pToS,StoP,indexPattern,indexString)

return False

else:

if letter!=string[indexString]:

# print (30,pToS,StoP,indexPattern,indexString)

return False

indexString+=1

# if maps is ok, check the nextone

return dfs(pToS,StoP,indexPattern+1,indexString)

else:

# we will have to scale the string to all possible that can map p

for i in range(indexString,len(string)):

potentialMap = string[indexString:i+1]

if potentialMap not in StoP: # it has to be somethign that was not map string to pattern

pToS[p] = potentialMap

StoP[potentialMap] = p

check = dfs(pToS,StoP,indexPattern+1,i+1)

# pop

pToS.pop(p)

StoP.pop(potentialMap)

if check:

return True

# print (46,pToS,StoP,indexPattern,indexString)

return False

else:

return False

return dfs({},{},0,0)

def wordPatternMatchAll(pattern: str, string: str) -> bool:

res = []

def dfs(pToS,StoP,indexPattern,indexString):

if indexPattern==len(pattern) and indexString==len(string):

temp = []

for key in pToS:

temp.append((key,pToS[key]))

res.append(temp)

elif indexPattern<len(pattern) and indexString<len(string):

# we can keep traverse

p = pattern[indexPattern]

if p in pToS:

#check if it maps out for the string

for letter in pToS[p]:

if indexString == len(string):

# print (26,pToS,StoP,indexPattern,indexString)

return False

else:

if letter!=string[indexString]:

# print (30,pToS,StoP,indexPattern,indexString)

return False

indexString+=1

# if maps is ok, check the nextone

return dfs(pToS,StoP,indexPattern+1,indexString)

else:

# we will have to scale the string to all possible that can map p

for i in range(indexString,len(string)):

potentialMap = string[indexString:i+1]

if potentialMap not in StoP: # it has to be somethign that was not map string to pattern

pToS[p] = potentialMap

StoP[potentialMap] = p

check = dfs(pToS,StoP,indexPattern+1,i+1)

# pop

pToS.pop(p)

StoP.pop(potentialMap)

if check:

return True

# print (46,pToS,StoP,indexPattern,indexString)

return False

else:

return False

dfs({},{},0,0)

return res

pattern = "abab"

string = "redblueredblue"

print (wordPatternMatchAll(pattern,string))

pattern = pattern = "aaaa"

string = "asdasdasdasd"

print (wordPatternMatchAll(pattern,string))

pattern = "aabb"

string = "xyzabcxzyabc"

print (wordPatternMatchAll(pattern,string))# -\*- coding: utf-8 -\*-

"""

Created on Tue Oct 8 02:02:25 2019

@author: Huy Nguyen

"""

from typing import List

#212. Word Search II

#Given a 2D board and a list of words from the dictionary, find all words in the board.

#

#Each word must be constructed from letters of sequentially adjacent cell, where "adjacent"

#cells are those horizontally or vertically neighboring. The same letter cell may not be used more than once in a word.

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

words=["baa","baae","aa"]

expected = ["abcdefg","befa","eaabcdgfa","gfedcbaaa","aa","aa"]

def findWords(board: List[List[str]], words: List[str]) -> List[str]:

myTrie= Trie(board,words)

return myTrie.res

class Trie:

def \_\_init\_\_(self,board,words):

self.d= {}

self.initialize(words)

self.res =self.search(board)

def isValidGrid(self,currentRow,currentCol,row,col):

return currentCol<col and currentRow<row and currentCol>=0 and currentRow >=0

def addWord(self,word):

root = self.d

for letter in word:

if letter not in root:

root[letter]={}

root = root[letter]

#need "#"

root["#"]=1

# root["?"] = 0 # have check this already

def initialize(self,words):

for word in words:

self.addWord(word)

def search(self,board):

row = len(board)

col = len(board[0])

root = self.d

visited = set()

res = set()

# store path to output a word

def dfs(currentRow,currentCol,visited,currentRoot,path):

if "#" in currentRoot: # we hit a word

res.add("".join(path))

self.prune(path)

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

for x,y in temp:

if self.isValidGrid(currentRow+x,currentCol+y,row,col) and (currentRow+x,currentCol+y) not in visited:

letter = board[currentRow+x][currentCol+y]

if letter in currentRoot:

# print (currentRow+x,currentRow+y,letter)

visited.add((currentRow+x,currentCol+y))

path.append(board[currentRow+x][currentCol+y])

dfs(currentRow+x,currentCol+y,visited,currentRoot[letter],path)

path.pop()

visited.remove((currentRow+x,currentCol+y))

for r in range(row):

for c in range(col):

letter = board[r][c]

if letter in root:

# print (65,r,c,letter)

visited.add((r,c))

path= [letter]

dfs(r,c,visited,root[letter],path)

visited.remove((r,c))

return res

def prune(self,path): # giving a currentRoot,check if we can prune up the path

root =self.d

def dfs(root,path,index):

letter = path[index]

nextRoot = root[letter]