**数算B必会简单题**

|  |  |
| --- | --- |
| **题目** | **tags** |
| 22782: PKU版爱消除 | stack |
| 26590: 检测括号嵌套 | stack |
| 26571: 我想完成数算作业：代码 | disjoint set |
| 20169: 排队 | disjoint set |
| 24744: 想要插队的Y君 | Linked List |
| 25143: 求二叉树的高度和叶子数目 | Tree |

# 001: PKU版爱消除

你有⼀个字符串S，⼤小写区分，⼀旦⾥⾯出现连续的PKU三个字符，就会消除。问最终稳定下来以后， 这个字符串是什么样的？

输入

⼀⾏,⼀个字符串S，表⽰消除前的字符串。 字符串S的⻓度不超过100000，且只包含⼤小写字⺟。

输出

⼀⾏,⼀个字符串T，表⽰消除后的稳定字符串 样例输入

TopSchoolPPKPPKUKUUPKUku

样例输出

TopSchoolPku

提示

请注意看样例。PKU消除后导致出现连续PKU，还要继续消除 比如APKPKUUB，消除中间PKU后，又得到PKU，就接着消除得到AB 此题用栈解决

s = input() stack = [] for c in s:

if c == "U":

if len(stack) >= 2 and stack[-1] == "K" and stack[-2] == "P": stack.pop()

stack.pop()

else:

stack.append(c)

else:

stack.append(c) print("".join(stack))

**002: 检测括号嵌套**

字符串中可能有3种成对的括号，"( )"、"[ ]"、"{}"。请判断字符串的括号是否都正确配对以及有无括号嵌 套。无括号也算正确配对。括号交叉算不正确配对，例如"1234[78)ab]"就不算正确配对。一对括号被包 含在另一对括号里面，例如"12(ab[8])"就算括号嵌套。括号嵌套不影响配对的正确性。 给定一个字符串: 如果括号没有正确配对，则输出 "ERROR" 如果正确配对了，且有括号嵌套现象，则输出"YES" 如果正确 配对了，但是没有括号嵌套现象，则输出"NO"

输入

一个字符串，长度不超过5000,仅由 ( ) [ ] { } 和小写英文字母以及数字构成 输出

根据实际情况输出 ERROR, YES 或NO 样例输入

样例1: [](){}

样例2:

[(a)]bv[]

样例3: [[(])]{}

样例输出

样例1: NO

样例2:

YES

样例3: ERROR

def check\_brackets(s): stack = []

nested = False

pairs = {')': '(', ']': '[', '}': '{'}

for ch in s:

if ch in pairs.values(): stack.append(ch)

elif ch in pairs.keys():

if not stack or stack.pop() != pairs[ch]: return "ERROR"

if stack:

nested = True

if stack:

return "ERROR"

return "YES" if nested else "NO"

s = input() print(check\_brackets(s))

# 003: 我想完成数算作业：代码

当卷王小D睡前意识到室友们每天熬夜吐槽的是自己也选了的课时，他距离早八随堂交的ddl只剩下了不 到4小时。已经debug一晚上无果的小D有心要分无力做题，于是决定直接抄一份室友的作业完事。万万 没想到，他们作业里完全一致的错误，引发了一场全面的作业查重……

假设a和b作业雷同，b和c作业雷同，则a和c作业雷同。所有抄袭现象都会被发现，且雷同的作业只有一 份独立完成的原版，请输出独立完成作业的人数

输入

第一行输入两个正整数表示班上的人数n与总比对数m，接下来m行每行均为两个1-n中的整数i和j，表明 第i个同学与第j个同学的作业雷同。

输出 独立完成作业的人数 样例输入

3 2

1 2

1 3

样例2：

4 2

2 4

1 3

样例输出

样例1：

1

样例2: 2

def find(parent, i):

if parent[i] != i:

parent[i] = find(parent, parent[i]) return parent[i]

def union(parent, x, y): xroot = find(parent, x) yroot = find(parent, y)

if xroot != yroot:

parent[xroot] = yroot

n, m =

parent

map(int, input().split())

=

for \_ in

i, j

list(range(n + 1)) range(m):

= map(int, input().split())

union(parent, i, j)

count = sum(i == parent[i] for i in range(1, n + 1)) print(count)

**004: 排队**

操场上有好多好多同学在玩耍，体育老师冲了过来，要求他们排队。同学们纪律实在太散漫了，老师不 得不来手动整队：

"A，你站在B的后面。"

"C，你站在D的后面。" "B，你站在D的后面。哦，去D队伍的最后面。"

更形式化地，初始时刻，操场上有 n 位同学，自成一列。每次操作，老师的指令是 "x y"，表示 x 所在的 队列排到 y 所在的队列的后面，即 x 的队首排在 y 的队尾的后面。（如果 x 与 y 已经在同一队列，请忽 略该指令） 最终的队列数量远远小于 n，老师很满意。请你输出最终时刻每位同学所在队列的队首（排 头），老师想记录每位同学的排头，方便找人。

输入

第一行一个整数 T (T≤5)，表示测试数据组数。 接下来 T 组测试数据，对于每组数据，第一行两个整数 n

和 m (n,m≤30000)，紧跟着 m 行每行两个整数

x 和 y (1≤x,y≤n)。 输出

共 T 行。 每行 n 个整数，表示每位同学的排头。

2

4 2

1 2

3 4

5 4

1 2

2 3

4 5

1 3

2 2 4 4

3 3 3 5 5

def getRoot(a):

if parent[a] != a:

parent[a] = getRoot(parent[a]) return parent[a]

def merge(a, b):

pa = getRoot(a) pb = getRoot(b) if pa != pb:

parent[pa] = parent[pb]

t = int(input()) for i in range(t):

n, m = map(int, input().split())

parent = [i for i in range(n + 10)] for i in range(m):

x, y = map(int, input().split()) merge(x, y)

for i in range(1, n + 1): print(getRoot(i), end=" ")

# 注意，一定不能写成 print(parent[i],end= " ")

# 因为只有执行路径压缩getRoot(i)以后，parent[i]才会是i的树根

print()

# 005: 想要插队的Y君

很遗憾，一意孤行的Y君没有理会你告诉他的饮食计划并很快吃完了他的粮食储备。 但好在他捡到了一张校园卡，凭这个他可以偷偷混入领取物资的队伍。 为了不被志愿者察觉自己是只猫，他想要插到队伍的最中央。（插入后若有偶数个元素则选取靠后的位 置）

于是他又找到了你，希望你能帮他修改志愿者写好的代码，在发放顺序的中间加上他的学号6。 你虽然不理解志愿者为什么要用链表来写这份代码，但为了不被发现只得在此基础上进行修改：

输入 一行，若干个整数，组成一个链表。 输出

一行，在链表中间位置插入数字6后得到的新链表

### 样例输入1 8 1 0 9 7 5

### 样例输入2

1 2 3

### 样例输出1

8 1 0 6 9 7 5

### 样例输出2 1 2 6 3

class Node:

def init (self, data, next=None): self.data, self.next = data, next

class LinkList:

def init (self): self.head = None

def initList(self, data): self.head = Node(data[0]) p = self.head

for i in data[1:]: node = Node(i) p.next = node

p = p.next

def insertCat(self):

#your code starts here

ptr =

total while

self.head

= 0

ptr total

ptr =

is not None:

+= 1

ptr.next

if total % 2 == pos = total

else:

pos = total ptr = self.head

0:

// 2

// 2 +

1

for i in range(pos-1): ptr = ptr.next

nd = Node(6) nd.next = ptr.next ptr.next = nd

########

def printLk(self): p = self.head while p:

print(p.data, end=" ") p = p.next

print()

lst = list(map(int,input().split())) lkList = LinkList() lkList.initList(lst) lkList.insertCat()

lkList.printLk()