数算B必会简单题

Updated 0020 GMT+8 Jun 4, 2024

2024 spring, Complied by Hongfei Yan

取自gw班，http://dsbpython.openjudge.cn/easyprbs/

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
25143/27638: 求二叉树的高度和叶子数目	tree
25155: 深度优先遍历一个无向图	dfs
22508: 最小奖金方案	topological sort

001: PKU版爱消除

http://dsbpython.openjudge.cn/easyprbs/001/

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的长度不超过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

来源

Chen Jiali

```
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

http://dsbpython.openjudge.cn/easyprbs/002/

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

- [ ] { } 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
```

```python
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
```

```python
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 = map(int, input().split())
parent = list(range(n + 1))
for _ in range(m):
    i, j = 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

http://cs101.openjudge.cn/practice/20169/

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。

样例输入

```
2
4 2
1 2
3 4
5 4
1 2
2 3
4 5
1 3
```

样例输出

```
2 2 4 4
3 3 3 5 5
```

```python
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

http://dsbpython.openjudge.cn/easyprbs/005/

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：

```
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):
// 在此处补充你的代码
########
    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()
```

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

```
### 样例输入1
8 1 0 9 7 5
### 样例输入2
1 2 3
```

样例输出

```
### 样例输出1
8 1 0 6 9 7 5
### 样例输出2
1 2 6 3
```

来源

Lou Yuke

```
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 = self.head
        total = 0
        while ptr is not None:
            total += 1
            ptr = ptr.next
        if total % 2 == 0:
            pos = total // 2
        else:
            pos = total // 2 + 1
        ptr = self.head
```

```
        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()
```

```
# 求二叉树的高度和叶子数目    2022-09-06 20:36:28
class BinaryTree:
    def __init__(self, data, left=None, right=None):
        self.data, self.left, self.right = data, left, right

    def addLeft(self, tree):  # tree是一个二叉树
        self.left = tree

    def addRight(self, tree):  # tree是一个二叉树
        self.right = tree

    def preorderTraversal(self, op):  # 前序遍历,对本题无用 op是函数，表示访问操作
        op(self)  # 访问根结点
        if self.left:  # 左子树不为空
            self.left.preorderTraversal(op)  # 遍历左子树
        if self.right:
            self.right.preorderTraversal(op)  # 遍历右子

    def inorderTraversal(self, op):  # 中序遍历， 对本题无用
        if self.left:
            self.left.inorderTraversal(op)
        op(self)
        if self.right:
            self.right.inorderTraversal(op)

    def postorderTraversal(self, op):  # 后序遍历， 对本题无用
        if self.left:
            self.left.postorderTraversal(op)
        if self.right:
            self.right.postorderTraversal(op)
        op(self)
```

```
    def bfsTraversal(self, op):  # 按层次遍历，对本题无用
        import collections
        dq = collections.deque()
        dq.append(self)
        while len(dq) > 0:
            nd = dq.popleft()
            op(nd)
            if nd.left:
                dq.append(nd.left)
            if nd.right:
                dq.append(nd.right)

    def countLevels(self):  # 算有多少层结点
        def count(root):
            if root is None:
                return 0
            return 1 + max(count(root.left), count(root.right))

        return count(self)

    def countLeaves(self):  # 算叶子数目
        def count(root):
            if root.left is None and root.right is None:
                return 1
            elif root.left is not None and root.right is None:
                return count(root.left)
            elif root.left is None and root.right is not None:
                return count(root.right)
            else:
                return count(root.right) + count(root.left)

        return count(self)

    def countWidth(self):  # 求宽度，对本题无用
        dt = {}

        def traversal(root, level):
            if root is None:
                return
            dt[level] = dt.get(level, 0) + 1
            traversal(root.left, level + 1)
            traversal(root.right, level + 1)

        traversal(self, 0)
        width = 0
        for x in dt.items():
            width = max(width, x[1])
        return width


def buildTree(n):
    nodes = [BinaryTree(None) for i in range(n)]
    isRoot = [True] * n
    # 树描述：  结点编号从0开始
```

```
        # 1 2
        # -1 -1
        # -1 -1
        for i in range(n):
            L, R = map(int, input().split())
            nd = i
            nodes[nd].data = nd
            if L != -1:
                nodes[nd].left = nodes[L]
                isRoot[L] = False
            if R != -1:
                nodes[nd].right = nodes[R]
                isRoot[R] = False
        for i in range(n):
            if isRoot[i]:
                return nodes[i]
        return None


n = int(input())
tree = buildTree(n)
print(tree.countLevels() - 1, tree.countLeaves())
```

006: 求二叉树的高度和叶子数目

http://cs101.openjudge.cn/dsapre/27638/

http://dsbpython.openjudge.cn/easyprbs/006/

给定一棵二叉树，求该二叉树的高度和叶子数目

二叉树高度定义：从根结点到叶结点依次经过的结点（含根、叶结点）形成树的一条路径，最长路径的结点数减1为树的高度。只有一个结点的二叉树，高度是0。

输入

第一行是一个整数n，表示二叉树的结点个数。二叉树结点编号从0到n-1。n <= 100 接下来有n行，依次对应二叉树的编号为0,1,2....n-1的节点。 每行有两个整数，分别表示该节点的左儿子和右儿子的编号。如果第一个（第二个）数为-1则表示没有左（右）儿子

输出

在一行中输出2个整数，分别表示二叉树的高度和叶子结点个数

样例输入

```
3
-1 -1
0 2
-1 -1
```

样例输出

```
1 2
```

来源

Guo Wei

由于输入无法分辨谁为根节点，所以写寻找根节点语句。

```
class TreeNode:
    def __init__(self):
        self.left = None
        self.right = None

def tree_height(node):
    if node is None:
        return -1  # 根据定义，空树高度为-1
    return max(tree_height(node.left), tree_height(node.right)) + 1

def count_leaves(node):
    if node is None:
        return 0
    if node.left is None and node.right is None:
        return 1
    return count_leaves(node.left) + count_leaves(node.right)

n = int(input())  # 读取节点数量
nodes = [TreeNode() for _ in range(n)]
has_parent = [False] * n  # 用来标记节点是否有父节点

for i in range(n):
    left_index, right_index = map(int, input().split())
    if left_index != -1:
        nodes[i].left = nodes[left_index]
        has_parent[left_index] = True
    if right_index != -1:
        #print(right_index)
        nodes[i].right = nodes[right_index]
        has_parent[right_index] = True

# 寻找根节点，也就是没有父节点的节点
root_index = has_parent.index(False)
root = nodes[root_index]

# 计算高度和叶子节点数
height = tree_height(root)
leaves = count_leaves(root)

print(f"{height} {leaves}")
```

注意：需要找根节点

```
class TreeNode:
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None


def build_tree(node_descriptions):
    nodes = {i: TreeNode(i) for i in range(len(node_descriptions))}
    child_set = set()

    for i, (left, right) in enumerate(node_descriptions):
        if left != -1:
            nodes[i].left = nodes[left]
            child_set.add(left)
        if right != -1:
            nodes[i].right = nodes[right]
            child_set.add(right)

    # Root is the node that is not anyone's child
    root = next(node for node in nodes.values() if node.val not in child_set)
    return root


def tree_height_and_leaf_count(root):
    if not root:
        return 0, 0  # height is 0 for empty tree, no leaves

    def dfs(node):
        if not node:
            return -1, 0

        if not node.left and not node.right:
            return 0, 1

        left_height, left_leaves = dfs(node.left)
        right_height, right_leaves = dfs(node.right)

        current_height = 1 + max(left_height, right_height)
        current_leaves = left_leaves + right_leaves

        return current_height, current_leaves

    height, leaf_count = dfs(root)
    return height, leaf_count


n = int(input())
node_descriptions = [tuple(map(int, input().split())) for _ in range(n)]
```

```
root = build_tree(node_descriptions)
height, leaf_count = tree_height_and_leaf_count(root)

print(height, leaf_count)
```

007: 深度优先遍历一个无向图

http://dsbpython.openjudge.cn/easyprbs/007/

输出无向图深度优先遍历序列

输入

第一行是整数n和m(0 < n <=16)，表示无向图有n个顶点，m条边，顶点编号0到n-1。接下来m行，每行两个整数a,b，表示顶点a,b之间有一条边。

输出

任意一个深度优先遍历序列

样例输入

```
9 9
0 1
0 2
3 0
2 1
1 5
1 4
4 5
6 3
8 7
```

样例输出

```
0 1 2 4 5 3 6 8 7
```

提示

题目需要Special Judge。所以输出错误答案也可能导致Runtime Error

来源

Guo Wei

```
def dfs(graph, visited, node):
    visited[node] = True
```

```
    print(node, end=" ")

    for neighbor in graph[node]:
        if not visited[neighbor]:
            dfs(graph, visited, neighbor)

def main():
    n, m = map(int, input().split())
    graph = [[] for _ in range(n)]
    visited = [False] * n

    for _ in range(m):
        a, b = map(int, input().split())
        graph[a].append(b)
        graph[b].append(a)

    for i in range(n):
        if not visited[i]:
            dfs(graph, visited, i)

if __name__ == "__main__":
    main()
```

```
def dfsTravel(G,op): #G是邻接表
    def dfs(v):
        visited[v] = True
        op(v)
        for u in G[v]:
            if not visited[u]:
                dfs(u)
    n = len(G)  # 顶点数目
    visited = [False for i in range(n)]
    for i in range(n):  # 顶点编号0到n-1
        if not visited[i]:
            dfs(i)

n,m = map(int,input().split())
G = [[] for i in range(n)]
for i in range(m):
    s,e = map(int,input().split())
    G[s].append(e)
    G[e].append(s)
dfsTravel(G,lambda x:print(x,end = " "))
```

008: 最小奖金方案

http://dsbpython.openjudge.cn/easyprbs/008/

现在有n个队伍参加了比赛，他们进行了m次PK。现在赛事方需要给他们颁奖（奖金为整数），已知参加比赛就可获得100元，由于比赛双方会比较自己的奖金，所以获胜方的奖金一定要比败方奖金高。请问赛事方要准备

的最小奖金为多少？奖金数额一定是整数。

输入

一组数据，第一行是两个整数n(1≤n≤1000)和m(0≤m≤2000)，分别代表n个队伍和m次pk，队伍编号从0到n-1。接下来m行是pk信息，具体信息a，b，代表编号为a的队伍打败了编号为b的队伍。 输入保证队伍之间的pk战胜关系不会形成有向环

输出

给出最小奖金w

样例输入

```
5 6
1 0
2 0
3 0
4 1
4 2
4 3
```

样例输出

```
505
```

来源

陈鑫

```
import collections
n,m = map(int,input().split())
G = [[] for i in range(n)]
award = [0 for i in range(n)]
inDegree = [0 for i in range(n)]

for i in range(m):
    a,b = map(int,input().split())
    G[b].append(a)
    inDegree[a] += 1
q = collections.deque()
for i in range(n):
    if inDegree[i] == 0:
        q.append(i)
        award[i] = 100
while len(q) > 0:
    u = q.popleft()
    for v in G[u]:
        inDegree[v] -= 1
```

```
            award[v] = max(award[v],award[u] + 1)
            if inDegree[v] == 0:
                q.append(v)
    total = sum(award)
    print(total)
```