#PKU 版爱消除　2022-06-05 00:26:52

```
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))
```

检测括号嵌套，想完成数算作业

```
s = input()
stack = ['A']
dt = {"]":"[",")":"(","}":"{"}
maxDepth = 0
found = False
for c in s:
    if c in '[({':
        stack.append(c)
    elif c in '])}':
        if stack[-1] == dt[c]:
            if len(stack) < maxDepth:
                found = True
            else:
                maxDepth = len(stack)
            stack.pop(-1)
        else:
            print("ERROR")
            exit()
if len(stack) > 1:
    print("ERROR")
else:
    if found:
        print("YES")
    else:
        print("NO")
```

```
n,m = map(int,input().split())
parent = [i for i in range(n+10)]
#sum = [1 for i in range(n+10)] #如果要统计每个
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]
        #sum[pb] += sum[pa] #如果要统计最终每个
        #则最后
for i in range(m):
    a,b = map(int,input().split())
    merge(a,b)
total = 0
for i in range(1,n+1):
    if parent[i] == i:  #只有父结点是自身的结点
        total += 1
print(total)
```

#如果要统计每个集合最终多少个元素，可以定义这个列表

#sum[pb] += sum[pa] #如果要统计最终每个集合的元素个数，可以开设 sum[]列表，此处执行本语句

#则最后

#只有父结点是自身的结点，才是树根。注意，只有 i 为树根时，sum[i]才能表示集合的元素个数

排队

```python
parent = None
def getRoot(a):
    if parent[a] != a:
        parent[a] = getRoot(parent[a])
    return parent[a]（路径压缩？）
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()
```

```python
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.nextS

    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 = n
# your code ends here
    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()
```

```python
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())
```

最小奖金方案和深度优先遍历一个无向图

```
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)
        ##这段看不明白
        if inDegree[v] == 0:
            q.append(v)
total = sum(award)
print(total)
```

```
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 = " "))
```

基本操作:

表 1-3　Python 列表提供的方法

方 法 名	用　法	解　释
append	alist.append(item)	在列表末尾添加一个新元素
insert	alist.insert(i,item)	在列表的第 *i* 个位置插入一个元素
pop	alist.pop()	删除并返回列表中最后一个元素
pop	alist.pop(i)	删除并返回列表中第 *i* 个位置的元素
sort	alist.sort()	将列表元素排序
reverse	alist.reverse()	将列表元素倒序排列
del	del alist[i]	删除列表中第 *i* 个位置的元素
index	alist.index(item)	返回 item 第一次出现时的下标
count	alist.count(item)	返回 item 在列表中出现的次数
remove	alist.remove(item)	从列表中移除第一次出现的 item

```
>>> myList
```

表 1-4　Python 字符串提供的方法

方 法 名	用　法	解　释
center	astring.center(w)	返回一个字符串，原字符串居中，使用空格填充新字符串，使其长度为 w
count	astring.count(item)	返回 item 出现的次数
ljust	astring.ljust(w)	返回一个字符串，将原字符串靠左放置并填充空格至长度 w
rjust	astring.rjust(w)	返回一个字符串，将原字符串靠右放置并填充空格至长度 w
lower	astring.lower()	返回均为小写字母的字符串
upper	astring.upper()	返回均为大写字母的字符串
find	astring.find(item)	返回 item 第一次出现时的下标
split	astring.split(schar)	在 schar 位置将字符串分割成子串

表 1-7　Python 字典支持的运算

运 算 名	运 算 符	解　释
[]	myDict[k]	返回与 k 相关联的值，如果没有则报错
in	key in adict	如果 key 在字典中，返回 True，否则返回 False
del	del adict[key]	从字典中删除 key 的键–值对

表 1-8　Python 字典提供的方法

方 法 名	用　法	解　释
keys	adict.keys()	返回包含字典中所有键的 dict_keys 对象
values	adict.values()	返回包含字典中所有值的 dict_values 对象
items	adict.items()	返回包含字典中所有键–值对的 dict_items 对象
get	adict.get(k)	返回 k 对应的值，如果没有则返回 None
get	adict.get(k, alt)	返回 k 对应的值，如果没有则返回 alt

```
print("%s is %d years old." % (aName, age))
```

这个简单的例子展示了一个新的字符串表达式。%是字符串运算符，被称作**格式化运算符**。表达式的左边部分是模板（也叫格式化字符串），右边部分则是一系列用于格式化字符串的值。需要注意的是，右边的值的个数与格式化字符串中%的个数一致。这些值将依次从左到右地被换入格式化字符串。

让我们更进一步地观察这个格式化表达式的左右两部分。格式化字符串可以包含一个或者多个转换声明。转换字符告诉格式化运算符，什么类型的值会被插入到字符串中的相应位置。在上面的例子中，%s 声明了一个字符串，%d 则声明了一个整数。其他可能的类型声明还包括 i、u、f、e、g、c 和%。表 1-9 总结了所有的类型声明。

表 1-9　格式化字符串可用的类型声明

字　符	输出格式
d、i	整数
u	无符号整数
f	m.dddd 格式的浮点数
e	m.dddde+/–xx 格式的浮点数
E	m.ddddE+/–xx 格式的浮点数
g	对指数小于–4 或者大于 5 的使用%e，否则使用%f
c	单个字符
s	字符串，或者任意可以通过 str 函数转换成字符串的 Python 数据对象
%	插入一个常量%符号

- ❑ enqueue(item)在队列的尾部添加一个元素。它需要一个元素作为参数，不返回任何值。
- ❑ dequeue()从队列的头部移除一个元素。它不需要参数，且会返回一个元素，并修改队列的内容。

```
deque()
    append()
    appendleft()
    extend()
    extendleft()
    pop()
    popleft()
    count()
    insert(index,obj)
```

链表制作：

代码清单 3-16　Node 类

```
1   class Node:
2       def __init__(self, initdata):
3           self.data = initdata
4           self.next = None
5
6       def getData(self):
7           return self.data
8
9       def getNext(self):
10          return self.next
11
12      def setData(self, newdata):
13          self.data = newdata
14
15      def setNext(self, newnext):
16          self.next = newnext
```

代码清单 3-17 UnorderedList 类的构造方法

```
class UnorderedList:
    def __init__(self):
        self.head = None
```

代码清单 3-19 add 方法

```
def add(self, item):
    temp = Node(item)
    temp.setNext(self.head)
    self.head = temp
```

代码清单 3-20 length 方法

```
def length(self):
    current = self.head
    count = 0
    while current != None:
        count = count + 1
        current = current.getNext()

    return count
```

代码清单 3-21 search 方法

```
def search(self, item):
    current = self.head
    found = False
    while current != None and not found:
        if current.getData() == item:
            found = True
        else:
            current = current.getNext()

    return found
```

代码清单 3-22 remove 方法

```
def remove(self, item):
    current = self.head
    previous = None
    found = False
    while not found:
        if current.getData() == item:
            found = True
        else:
            previous = current
            current = current.getNext()

    if previous == None:
        self.head = current.getNext()
    else:
        previous.setNext(current.getNext())
```

有序列表：

代码清单 3-24　有序列表的 search 方法

```
1   def search(self, item):
2       current = self.head
3       found = False
4       stop = False
5       while current != None and not found and not stop:
6           if current.getData() == item:
7               found = True
8           else:
9               if current.getData() > item:
10                  stop = True
11              else:
12                  current = current.getNext()
13
14      return found
```

代码清单 3-25　有序列表的 add 方法

```
1   def add(self, item):
2       current = self.head
3       previous = None
4       stop = False
5       while current != None and not stop:
6           if current.getData() > item:
7               stop = True
8           else:
9               previous = current
10              current = current.getNext()
11
12      temp = Node(item)
13      if previous == None:
14          temp.setNext(self.head)
15          self.head = temp
16      else:
17          temp.setNext(current)
18          previous.setNext(temp)
```

列表排序:

```
list.sort(cmp=None, key=None, reverse=False)
```

字典排序:

```
8  # 根据key的升序排列，把key value都打印出来
9  new_sys1 = sorted(sys.items(), key=lambda d: d[0], reverse=False)
10 print(new_sys1)
11
12 new_sys1 = sorted(sys.items(), reverse=False)
13 print(new_sys1)
```

印结果:

```
1  ['age', 'gender', 'name']
2  ['age', 'gender', 'name']
3  [('age', '十八'), ('gender', 'man'), ('name', '张三')]
4  [('age', '十八'), ('gender', 'man'), ('name', '张三')]
5
```

根据字典的value值进行排序

```
1  # 单独打印出排序后的value值
2  new_sys1 = sorted(sys.values())
3  print(new_sys1)
4
5  # 打印出根据value排序后的键值对的具体值
6  new_sys2 = sorted(sys.items(),  key=lambda d: d[1], reverse=False)
7  print(new_sys2)
```