**Assignment #4: 排序、栈、队列和树**

Updated 0005 GMT+8 March 11, 2024

2024 spring, Complied by **王申睿——物理学院**

**说明：**

1）The complete process to learn DSA from scratch can be broken into 4 parts:

Learn about Time complexities, learn the basics of individual Data Structures, learn the basics of Algorithms, and practice Problems.

2）请把每个题目解题思路（可选），源码Python, 或者C++ （已经在Codeforces/Openjudge上AC），截图（包含 Accepted），填写到下面作业模版中（推荐使用 typora[https://typoraio.cn](https://typoraio.cn/) ，或者用word）。 AC 或者没有AC ， 都请标上每个题目大致花费时间。

3）提交时候先提交pdf文件，再把md或者doc文件上传到右侧 “作业评论” 。Canvas需要有同学清晰头像、提交文 件有pdf、"作业评论" 区有上传的md或者doc附件。

4）如果不能在截止前提交作业，请写明原因。

**编程环境**

（请改为同学的操作系统、编程环境等）

操作系统： macOS Ventura 13.4.1 (c)

Python编程环境： Spyder IDE 5.2.2, PyCharm 2023.1.4 (Professional Edition)

C/C++编程环境： Mac terminal vi (version 9.0.1424), g++/gcc (Apple clang version 14.0.3, clang- 1403.0.22.14.1)

**1. 题目**

**05902: 双端队列**

<http://cs101.openjudge.cn/practice/05902/>

思路：

代码

from collections import deque

t=int(input())

for \_ in range(t):

n=int(input())

a=deque()

for \_ in range(n):

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

if b==1:

a.append(c)

elif b==2:

if c==0:

a.popleft()

elif c==1:

a.pop()

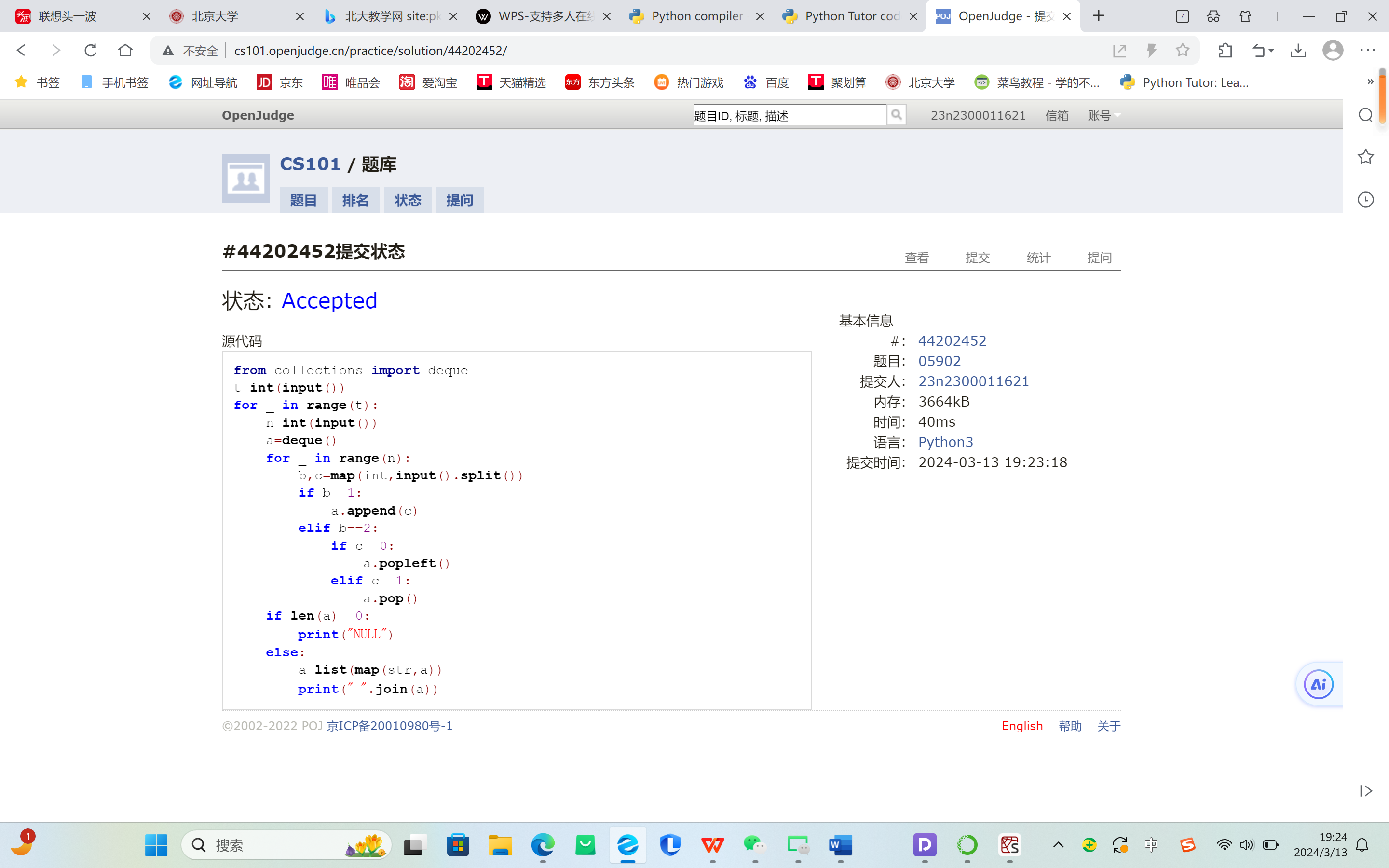
if len(a)==0:

print("NULL")

else:

a=list(map(str,a))

print(" ".join(a))



**02694: 波兰表达式**

<http://cs101.openjudge.cn/practice/02694/>

思路：

代码

a=list(map(str,input().split()))

t=0

b=["+","-","\*","/"]

while len(a)>=3:

if a[t]=="+":

if a[t+1] not in b and a[t+2] not in b:

c=float(a[t+1])+float(a[t+2])

a[t:t+3]=[c]

elif a[t]=="-":

if a[t+1] not in b and a[t+2] not in b:

c=float(a[t+1])-float(a[t+2])

a[t:t+3]=[c]

elif a[t]=="\*":

if a[t+1] not in b and a[t+2] not in b:

c=float(a[t+1])\*float(a[t+2])

a[t:t+3]=[c]

elif a[t]=="/":

if a[t+1] not in b and a[t+2] not in b:

c=float(a[t+1])/float(a[t+2])

a[t:t+3]=[c]

if t<len(a)-3:

t+=1

else:

t=0

print("{:.6f}".format(float(a[0])))



**24591: 中序表达式转后序表达式**

<http://cs101.openjudge.cn/practice/24591/>

思路：

代码

1 #

2

（已学习调度场算法）

**22068: 合法出栈序列**

<http://cs101.openjudge.cn/practice/22068/>

思路：

代码

（因为思路不对，所以借鉴了题解）

def is\_valid\_pop\_sequence(origin, output):

if len(origin) != len(output):

return False # 长度不同，直接返回False

stack = []

bank = list(origin)

for char in output:

# 如果当前字符不在栈顶，且bank中还有字符，则继续入栈

while (not stack or stack[-1] != char) and bank:

stack.append(bank.pop(0))

# 如果栈为空，或栈顶字符不匹配，则不是合法的出栈序列

if not stack or stack[-1] != char:

return False

stack.pop() # 匹配成功，弹出栈顶元素

return True # 所有字符都匹配成功

# 读取原始字符串

origin = input().strip()

# 循环读取每一行输出序列并判断

while True:

try:

output = input().strip()

if is\_valid\_pop\_sequence(origin, output):

print('YES')

else:

print('NO')

except EOFError:

break

（这是我自己的尝试，能够应付所有小数据量样例）

from collections import deque

x=deque(input())

k=len(x)

s=[]

t=[]

m=set()

def hefa(n):

global x

if n==0:

s.append(x.popleft())

elif n==1:

t.append(s.pop(-1))

if s and not x:

hefa(1)

s.append(t.pop(-1))

elif x and not s:

hefa(0)

x.appendleft(s.pop(-1))

elif x and s:

hefa(1)

s.append(t.pop(-1))

hefa(0)

x.appendleft(s.pop(-1))

else:

m.add("".join(t))

return

hefa(0)

while True:

try:

p=input()

if len(p)!=k:

print("NO")

elif p in m:

print("YES")

else:

print("NO")

except EOFError:

break

代码运行截图

（AC代码截图，至少包含有"Accepted"）

**06646: 二叉树的深度**

<http://cs101.openjudge.cn/practice/06646/>

思路：

代码

class TreeNode:

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

self.left=None

self.right=None

def jiedianshu(node):

if node is None:

return 0

if node.left is None and node.right is None:

return 1

return max(jiedianshu(node.left),jiedianshu(node.right))+1

n=int(input())

nodes=[TreeNode() for \_ in range(n)]

for i in range(n):

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

if zuo!=-1:

nodes[i].left=nodes[zuo-1]

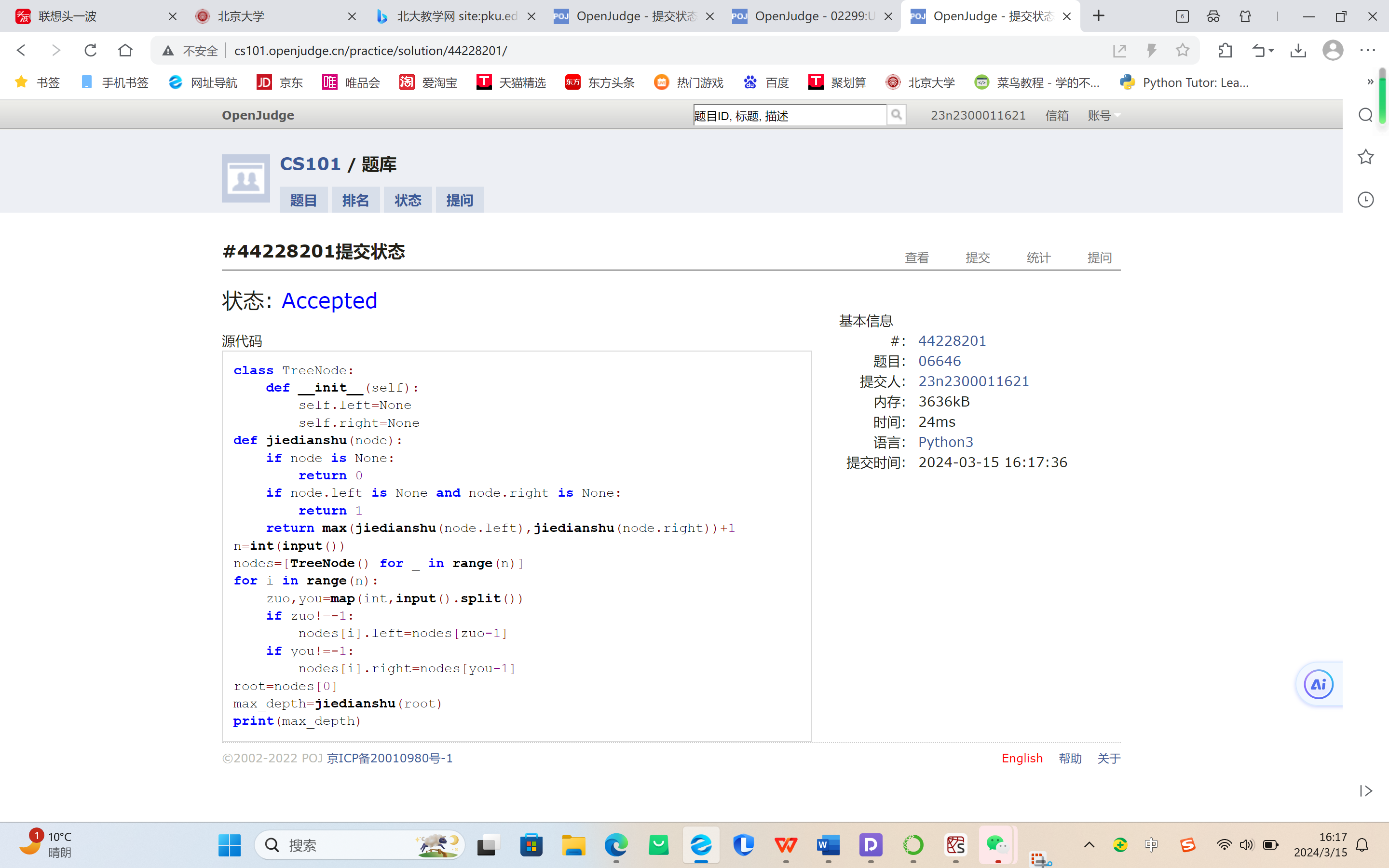
if you!=-1:

nodes[i].right=nodes[you-1]

root=nodes[0]

max\_depth=jiedianshu(root)

print(max\_depth)



**02299: Ultra-QuickSort**

<http://cs101.openjudge.cn/practice/02299/>

思路：

代码

def fenyepaixu(s):

m=len(s)

if m<2:

return s,0

mid=m//2

s1=s[:mid]

s2=s[mid:]

s1,count1=fenyepaixu(s1)

s2,count2=fenyepaixu(s2)

s,count3=count(s1,s2)

return s,count1+count2+count3

def count(b,c):

i=j=0

res=[]

count=0

while i<len(b) and j<len(c):

if b[i]<=c[j]:

res.append(b[i])

i+=1

else:

res.append(c[j])

count+=len(b)-i

j+=1

res=res+b[i:]+c[j:]

return res,count

while True:

n=int(input())

a=[]

if n==0:

break

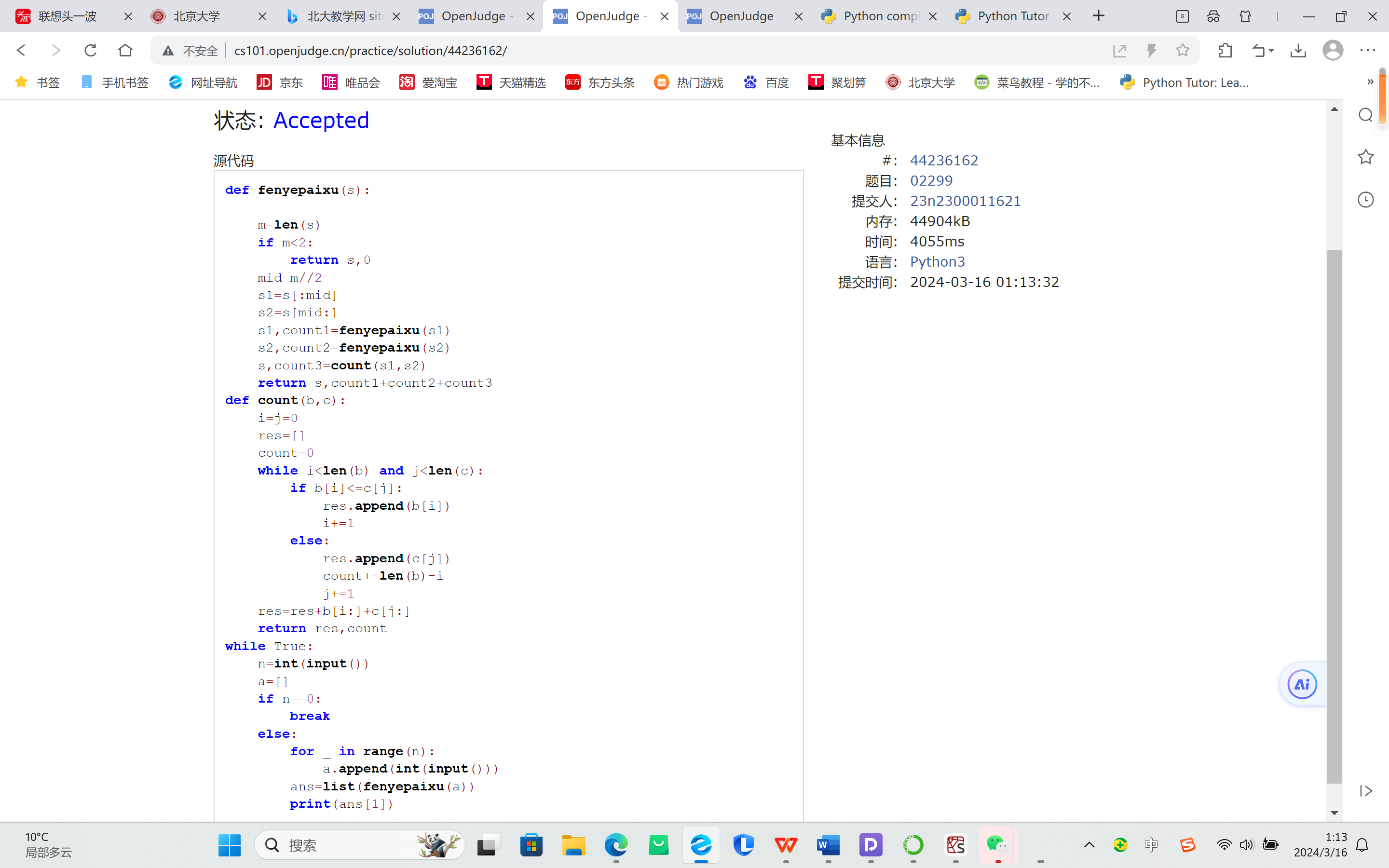
else:

for \_ in range(n):

a.append(int(input()))

ans=list(fenyepaixu(a))

print(ans[1])



**2. 学习总结和收获**

上学期在做cheatsheet的时候记下了双端队列，这次有幸体验了一把这个collections库里的宝藏。作为Python的内置的对象，它比普通列表的pop方法要快。这种数据结构的本质是链表，而Python的底层语言C语言又十分擅长指针，所以用双端队列做只用了区区40ms。

本次借课件中的代码学习了树的构建：第一步，建立树节点类，属性都是子节点；第二步，创建一个包含树节点实例的列表；第三步，根据输入提供的子节点编号修改每个节点的子节点属性的值，赋值后的子节点属性的值不为None。

但是，这样的构建方式的不足是只能用于二叉树，而无法得到更多子节点的树。理论的和方法的统一、普适性是物理人的毕生追求，所以我必须要突破这个限制。对于这个问题的解决方法，GPT-3.5给出了如下的方案：

class TreeNode:

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

self.value = value

self.children = []

def add\_child(self, child\_node):

self.children.append(child\_node)

# 创建树的节点

root = TreeNode("A")

child1 = TreeNode("B")

child2 = TreeNode("C")

child3 = TreeNode("D")

# 添加子节点

root.add\_child(child1)

root.add\_child(child2)

root.add\_child(child3)

# 为子节点添加子节点

child1.add\_child(TreeNode("E"))

child1.add\_child(TreeNode("F"))

child2.add\_child(TreeNode("G"))

# 打印树的结构

def print\_tree(node, level=0):

print(" " \* level + node.value)

for child in node.children:

print\_tree(child, level + 1)

print\_tree(root)

根据以上方法，我们可以对课件中的类进行修改，使其能够适应几乎所有的情况：

class TreeNode():

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

self.value=value

self.children=[]

def add\_child(self,child):

self.children.append(child)

def depth(node):

if node is None:

return 0

if not node.children:

return 1

return max(depth(i) for i in node.children)+1

def height(node):

if node is None:

return -1

return max(height(i) for i in node.children)+1

def jiedianshu(node):

if node is None:

return 0

if not node.children:

return 1

return sum(jiedianshu(j) for j in node.children)+1

#以上所有的+1都是为了拥有子节点时将作为参数的父节点囊括进去

n=int(input())

nodes=[TreeNode(None) for \_ in range(n)]

for i in range(n):

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

if not all(j==-1 for j in m):

nodes[i].value=i

for j in m:

if j!=-1:

nodes[i].add\_child(nodes[j-1])

root=nodes[0]

max\_depth=depth(root)

print(max\_depth)



经检验，新方法完全可以适应二叉树的要求，还能处理任意数量子节点的情况，具有更广的普适性，是一种更高级的方法。

Merge Sort（归并排序）是一种用递归分解问题的排序方式。其基本步骤是：1：将列表分割至不可再分；2：将分割的列表内部进行排序；3：在合并的过程中微调顺序，最终组装成整体。该方法跳过了大范围的比较，缩小了检验范围，从而有高概率降低时间复杂度。