Assignment #9: 图论：遍历，及 树算

Updated 1739 GMT+8 Apr 14, 2024

2024 spring, Complied by 城环 吴至超

说明：

1）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 ，或者用word）。AC 或者没有AC，都请标上每个题目大致花费时间。

2）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。

3）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：windows11

Python编程环境：pycharm2023.2.3

1. 题目

04081: 树的转换

http://cs101.openjudge.cn/dsapre/04081/

思路：根据输入递归先建立一般的树，并测量高度（某条路最多节点-1）

然后把一般的树转化为二叉树，思路是观察，某一节点若有子节点，则左-右-右-右...这样接上，左儿子代表儿子，右儿子代表兄弟，再测量新二叉树的高度

代码

```
#
#04081:树的转换
class  tree:
    def __init__(self,name):
        self.left=None
        self.right=None
        self.name=name
        self.children=[]
sample=list(input())
```

```
na=0
ni=0
root=tree(na)

def convert(root):#先构建一棵一般的树
    global na,ni
    while sample and sample.pop(0)=="d":
        na+=1
        newtree=tree(na)
        root.children.append(convert(newtree))
    return root

def measure(root):
    cnt=0
    if root:
        for i in root.children:
            cnt=max(1+measure(i),cnt)
        if not root.children:
            return 0
    return cnt

def buildleft(root):
    new=root.children[0]
    for i in range(1,len(root.children)):
        new.right=root.children[i]
        new=root.children[i]
    return root.children[0]#如何把根节点删掉

def construct(root):
    if root.children:
        root.left=buildleft(root)
        for m in root.children:
            construct(m)
    return root
def findheight(root):
    cnt=1
    if root.left and root.right:
        return cnt+max(findheight(root.left),findheight(root.right))
    elif root.left and not root.right:
        return cnt+findheight(root.left)
    elif root.right and not root.left:
        return cnt+findheight(root.right)
    else:
        return 1
    return cnt
height1=measure(convert(root))

height2=findheight(construct(convert(root)))-1#需要返回新的树根

print(f"{height1} => {height2}")
```

代码运行截图 ==（至少包含有"Accepted"）==

08581: 扩展二叉树

http://cs101.openjudge.cn/dsapre/08581/

思路：结合中序与叶子节点结束标志来递归建树

代码

```
#
#08581:扩展二叉树
class tree:
    def __init__(self,name):
        self.name=name
        self.left=None
        self.right=None
preorder=list(input())
root=tree(preorder.pop(0))

def buildtree(root):
    if preorder:
        newtree=tree(preorder.pop(0))
        if newtree.name!=".":
            root.left=buildtree(newtree)
        else:
            root.left= None
    if preorder:
        newtree = tree(preorder.pop(0))
        if newtree.name != ".":
            root.right = buildtree(newtree)
        else:
```

```
            root.right= None
    return root
def midorder(root):#中序，左中右
    stack=[]
    if not root:
        return
    if root.left:
        stack.extend(midorder(root.left))
    stack.append(root.name)
    if root.right:
        stack.extend(midorder(root.right))
    return stack

def postorder(root):
    stack=[]
    if not root:
        return
    if root.left:
        stack.extend(postorder(root.left))
    if root.right:
        stack.extend(postorder(root.right))
    stack.append(root.name)
    return stack

print("".join(midorder(buildtree(root))))
print("".join(postorder(buildtree(root))))
```

代码运行截图 ==（至少包含有"Accepted"）==

题目 排名 状态 提问

#44720703提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
#08581:扩展二叉树
class tree:
    def __init__(self,name):
        self.name=name
        self.left=None
        self.right=None
preorder=list(input())
root=tree(preorder.pop(0))

def buildtree(root):
    if preorder:
        newtree=tree(preorder.pop(0))
        if newtree.name!=".":
            root.left=buildtree(newtree)
        else:
            root.left= None
    if preorder:
        newtree = tree(preorder.pop(0))
        if newtree.name != ".":
            root.right = buildtree(newtree)
        else:
            root.right= None
    return root
def midorder(root):#中序，左中右
    stack=[]
    if not root:
        return
    if root.left:
        stack.extend(midorder(root.left))
    stack.append(root.name)
    if root.right:
        stack.extend(midorder(root.right))
    return stack

def postorder(root):
    stack=[]
    if not root:
        return
```

基本信息

#: 44720703
题目: 08581
提交人: 23n2300013289
内存: 4424kB
时间: 27ms
语言: Python3
提交时间: 2024-04-20 15:48:03

22067: 快速堆猪

http://cs101.openjudge.cn/practice/22067/

思路：字典懒删除标记+堆的利用，堆中所有元素都时刻在堆里，不会被弹出

代码

```
#
#22067:快速堆猪
import heapq
from collections import defaultdict
dic=defaultdict(int)#这样对于任意查询的键，其默认值均为int（0）
pigsheap=[]#用来判断最轻，很快
pigs=[]#用来表示堆积顺序
while True:
    try:
        a=input()
    except EOFError:
        break
    if a=="pop":#怎么在堆结构里把它踢掉
        if pigs:
            dic[pigs.pop()]+=1#0表示猪还在堆里，1表示猪已被踢出，因为在堆里remove的时间代价太大
    elif a=="min":
        if pigs:
            while True:#因为猪堆里有猪，这个循环保证了一定能输出最轻的猪重
                tempo=heapq.heappop(pigsheap)
                if not dic[tempo]:#表明猪还在栈里
                    heapq.heappush(pigsheap,tempo)#放回去，min不要求踢猪
                    print(tempo)
                    break
                dic[tempo]-=1#把踢出的已经不存在的猪还原为0，以便相同重量的猪加入时初始值为0
    elif a[0:2]=="pu":
        m=(a.split())
        heapq.heappush(pigsheap,int(m[1]))
        pigs.append(int(m[1]))
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

04123: 马走日

dfs, http://cs101.openjudge.cn/practice/04123

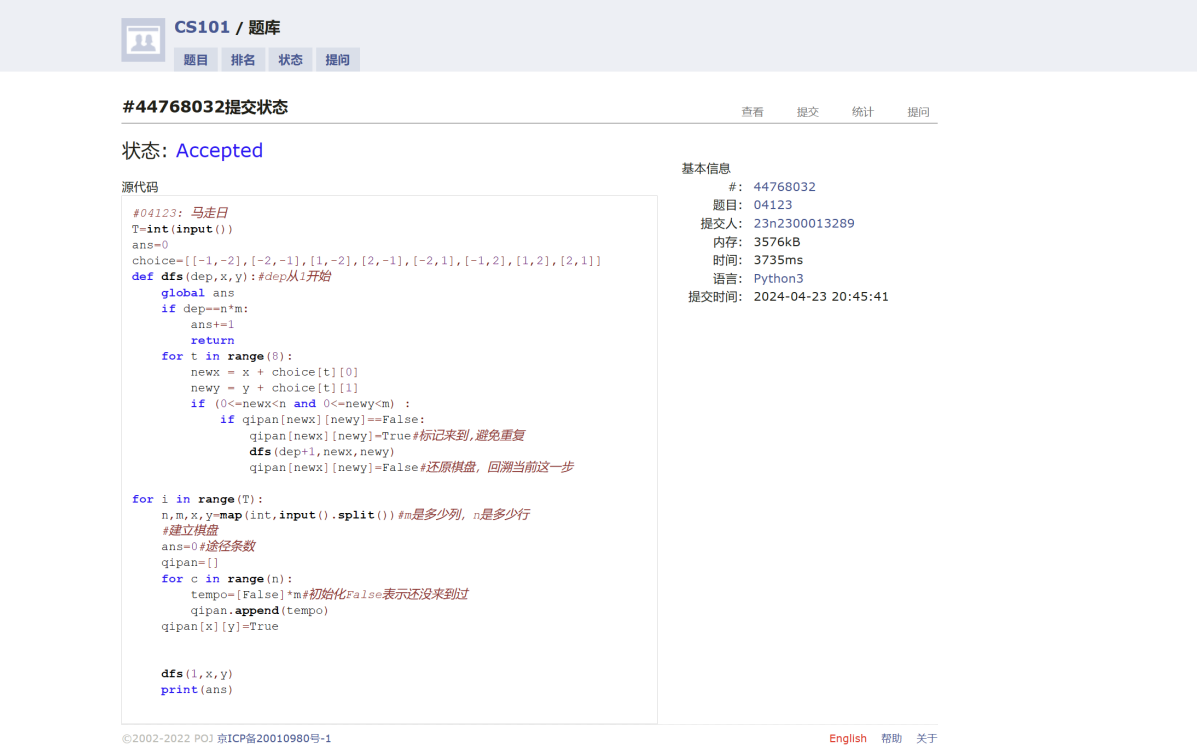
思路：dfs最大深度搜索

代码

```
#
#04123: 马走日
T=int(input())
ans=0
choice=[[-1,-2],[-2,-1],[1,-2],[2,-1],[-2,1],[-1,2],[1,2],[2,1]]
def dfs(dep,x,y):#dep从1开始
    global ans
    if dep==n*m:
        ans+=1
        return
    for t in range(8):
        newx = x + choice[t][0]
        newy = y + choice[t][1]
        if (0<=newx<n and 0<=newy<m) :
            if qipan[newx][newy]==False:
                qipan[newx][newy]=True#标记来到,避免重复
                dfs(dep+1,newx,newy)
                qipan[newx][newy]=False#还原棋盘，回溯当前这一步

for i in range(T):
    n,m,x,y=map(int,input().split())#m是多少列，n是多少行
    #建立棋盘
    ans=0#途径条数
```

```
    qipan=[]
    for c in range(n):
        tempo=[False]*m#初始化False表示还没来到过
        qipan.append(tempo)
    qipan[x][y]=True


    dfs(1,x,y)
    print(ans)
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==



28046: 词梯

bfs, http://cs101.openjudge.cn/practice/28046/

思路：利用桶（相差一个字母差异为桶的分类）来建立图

两种类的写法

回溯与最大宽度搜索bfs

重点在于标记previous

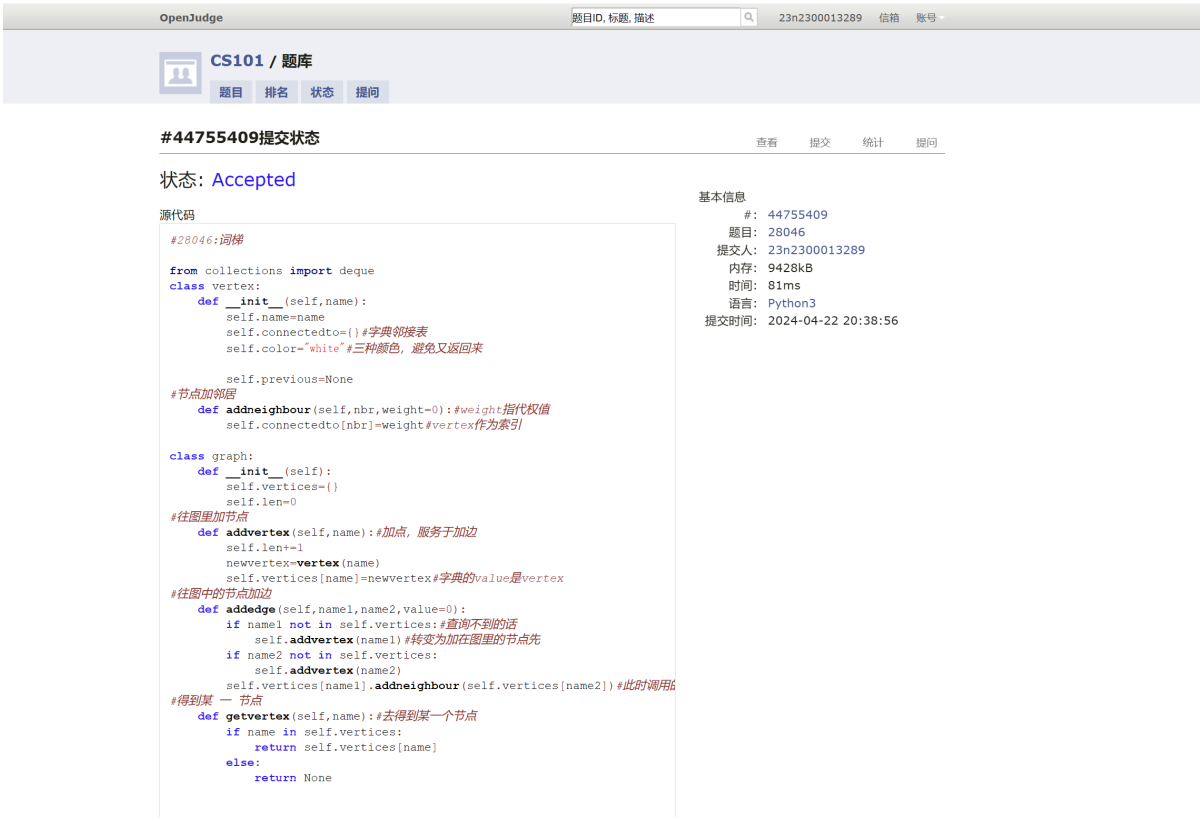
代码

```
#
#28046:词梯
```

```
from collections import deque
class vertex:
    def __init__(self,name):
        self.name=name
        self.connectedto={}#字典邻接表
        self.color="white"#三种颜色，避免又返回来

        self.previous=None
#节点加邻居
    def addneighbour(self,nbr,weight=0):#weight指代权值
        self.connectedto[nbr]=weight#vertex作为索引

class graph:
    def __init__(self):
        self.vertices={}
        self.len=0
#往图里加节点
    def addvertex(self,name):#加点，服务于加边
        self.len+=1
        newvertex=vertex(name)
        self.vertices[name]=newvertex#字典的value是vertex
#往图中的节点加边
    def addedge(self,name1,name2,value=0):
        if name1 not in self.vertices:#查询不到的话
            self.addvertex(name1)#转变为加在图里的节点先
        if name2 not in self.vertices:
            self.addvertex(name2)
        self.vertices[name1].addneighbour(self.vertices[name2])#此时调用的是节点的函
数，索引是vertex
#得到某 一 节点
    def getvertex(self,name):#去得到某一个节点
        if name in self.vertices:
            return self.vertices[name]
        else:
            return None
def buildgraph(arr):
    #先建桶
    buckets={}
    graph1=graph()
    for m in arr:
        for i,_ in enumerate(m):
            bucketname=m[:i]+"_"+m[i+1::]
            if bucketname not in buckets:
                buckets[bucketname]=set()
            buckets[bucketname].add(m)
    #然后建图
    for similiarity in buckets.values():#直接遍历字典的values值
        for i in similiarity:
            for m in similiarity-{i}:#set减set
                graph1.addedge(i,m)
    return graph1

def bfs(start,end):#从上往下开始找，start是起始单词的vertex版本
    queue=deque()#建立队列
    queue.append(start)
```

```
    while len(queue)>0:
        current=queue.popleft()#deque库的语句，与pop相区别
        if current==end:
            return True
        for m in current.connectedto:#遍历key值,赋值previous便于回溯
            if m.color=="white":#表明还没有previous
                m.color="black"
                m.previous=current
                queue.append(m)
        current.color="black"
    return False
def traverse(end):
    ans=[]
    current=end
    while current.previous:
        ans.append(current.name)
        current=current.previous
    ans.append(current.name)
    return ans

n=int(input())
arr=[]
for i in range(n):
    a=input()
    arr.append(a)
start,end=input().split()
Graph=buildgraph(arr)
start=Graph.getvertex(start)
end=Graph.getvertex(end)
if bfs(start,end):
    bfs(start,end)
    ans = traverse(end)
    print(" ".join(ans[::-1]))
else:
    print("NO")
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==



28050: 骑士周游

dfs, http://cs101.openjudge.cn/practice/28050/

思路：两种类+优化（先苦后甜类似围棋）+dfs深搜

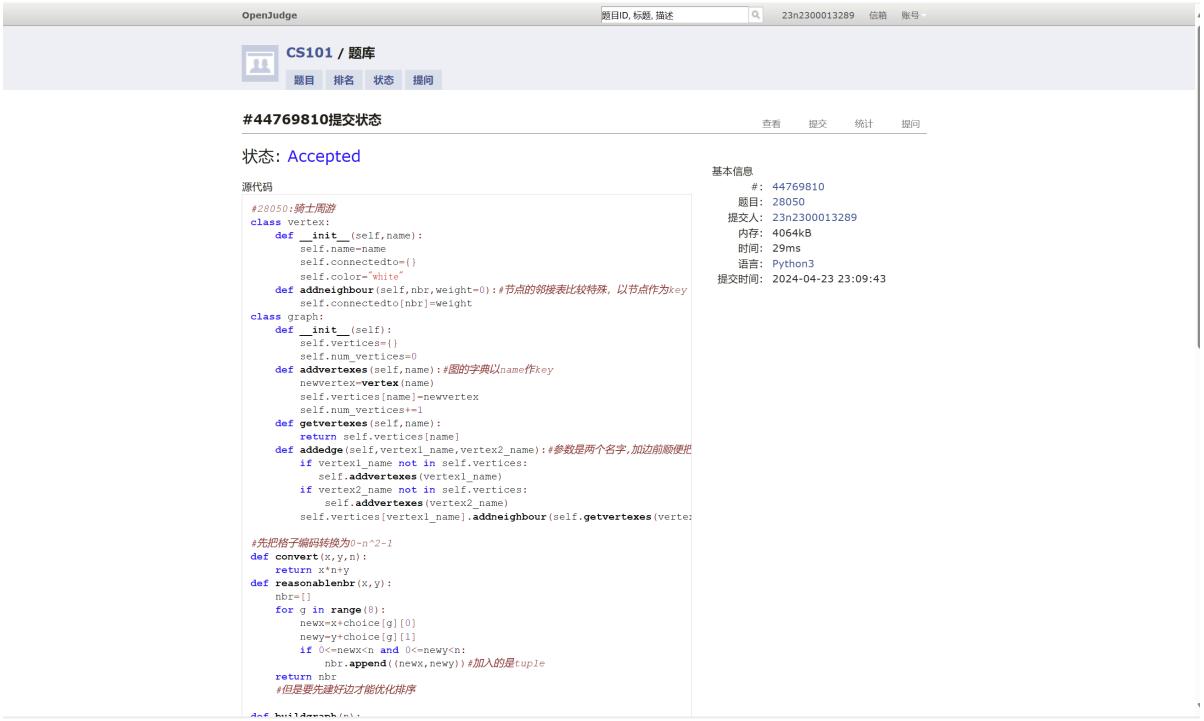
代码

```
#
#28050:骑士周游
class vertex:
    def __init__(self,name):
        self.name=name
        self.connectedto={}
        self.color="white"
    def addneighbour(self,nbr,weight=0):#节点的邻接表比较特殊，以节点作为key
        self.connectedto[nbr]=weight
class graph:
    def __init__(self):
        self.vertices={}
        self.num_vertices=0
    def addvertexes(self,name):#图的字典以name作key
        newvertex=vertex(name)
        self.vertices[name]=newvertex
        self.num_vertices+=1
    def getvertexes(self,name):
        return self.vertices[name]
```

```
    def addedge(self,vertex1_name,vertex2_name):#参数是两个名字,加边前顺便把节点加到字典
里
        if vertex1_name not in self.vertices:
           self.addvertexes(vertex1_name)
        if vertex2_name not in self.vertices:
            self.addvertexes(vertex2_name)
        self.vertices[vertex1_name].addneighbour(self.getvertexes(vertex2_name))

#先把格子编码转换为0~n^2-1
def convert(x,y,n):
    return x*n+y
def reasonablenbr(x,y):
    nbr=[]
    for g in range(8):
        newx=x+choice[g][0]
        newy=y+choice[g][1]
        if 0<=newx<n and 0<=newy<n:
            nbr.append((newx,newy))#加入的是tuple
    return nbr
    #但是要先建好边才能优化排序

def buildgraph(n):
    Graph=graph()
    for i in range(n):
        for m in range(n):
            new_num=convert(i,m,n)
            neighbour1=reasonablenbr(i,m)
            for t in neighbour1:
                new_num1=convert(t[0],t[1],n)    #得到了两个节点的name
                Graph.addedge(new_num,new_num1)
    return Graph

#建好图以后优化排序
def better_series(vertex):
    postseries=[]
    tempo=vertex.connectedto.keys()#一个列表
    for ii in tempo:
        if ii.color=="white":#如果还能踩
            cnt=0
            for iii in ii.connectedto.keys():
                if iii.color=="white":
                    cnt+=1
            postseries.append((cnt,ii))
    postseries.sort(key=lambda x:x[0])
    return [x[1] for x in postseries]
ans=0

def dfs(start,dep,limit,path):
    start.color="black"
    path.append(start)
    if dep<limit:

        neighbors=better_series(start)
        for nbr in neighbors:
            if nbr.color=="white" and dfs(nbr,dep+1,limit,path):
                return True
```

```
            else:
                path.pop()
                start.color="white"
                return False
    else:
        return True


choice=[[-1,-2],[-2,-1],[1,-2],[2,-1],[-2,1],[-1,2],[1,2],[2,1]]
n=int(input())
x,y=map(int,input().split())
Graph=buildgraph(n)
final=Graph.getvertexes(convert(x,y,n))
tourpath=[]
done=dfs(final,0,n**2-1,tourpath)
if done:
    print("success")
else:print("fail")
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==



2. 学习总结和收获

==如果作业题目简单，有否额外练习题目，比如：OJ“2024spring每日选做”、CF、LeetCode、洛谷等网站题目。==

树的镜面映射（补上周）

思路：递归+观察建树，和本次的第一题比较像

代码

```
class tree:
    def __init__(self,name):
        self.name=name
        self.left=None
        self.right=None
        self.children=[]

def buildtree(root,vertexes):#根据前序遍历与是否为叶子节点的判断建立伪满二叉树
    if root.name[0]=="$" or root.name[1]=="1":
        return root
    if root.left==None and vertexes:
        new = tree(vertexes.pop(0))
        root.left=buildtree(new,vertexes)
    if root.right==None and vertexes:
        new=tree(vertexes.pop(0))
        root.right=buildtree(new,vertexes)
    return root

def findright(root):
    stack=[root]
    if root.right and root.right.name!="$1":
        stack.extend(findright(root.right))
    return stack
def huanyuan(root): #自变量是伪满 二 叉树的根，企图将其还原为原镜像树，
     if root.name[1]!= "1" and root.left.name[1]!="1":#非叶子节点
         root.children=findright(root.left)
     elif root.name[1]!= "1" and root.left.name[1]=="1" and
root.left.name[0]!="$":
         root.children=[root.left]

     for i in root.children:
         huanyuan(i)
     return root

def mirror(root):#已核对
    if root.children:
        root.children=root.children[::-1]
        for m in root.children:
            mirror(m)
    return root

def bfs(root):#即层次遍历
    queue=[root]
    for i in queue:
        for m in i.children:
            queue.append(m)
    return queue
n=int(input())
vertexes=[x for x in input().split()]
root=tree(vertexes.pop(0))
priroot=mirror(huanyuan(buildtree(root,vertexes)))
prilis1=bfs(priroot)

for m in prilis1:
    print(m.name[0],end=" ")
print()
```

累死我了。发现自己已经可以较好地解决树的非实际应用问题了，重点是观察建树的规律。

但是图的有关设计还不太熟练

bfs与dfs学计概的时候没接触过，马走日还好，代码短很快就看明白了，这个骑士周游真的恶心，这么长的代码还要优化，好耗时间，但是逐渐对图的写法熟悉起来了。。。初学只能参考题解

词梯嘛，我看题目里的图片应该有不止一种答案吧？不太清楚是写法导致最后输出一种就行还是题目输入设计本身的缺陷。我觉得bfs能找到最短路径这点还是很奇妙的。