# Assignment #9: 图论：遍历，及 树算

Updated 1739 GMT+8 Apr 14, 2024

2024 spring, Complied by 夏天、生命科学学院

**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：Windows 10 家庭版

Python编程环境：Spyder （python 3.11）

# 1. 题目

## 04081: 树的转换

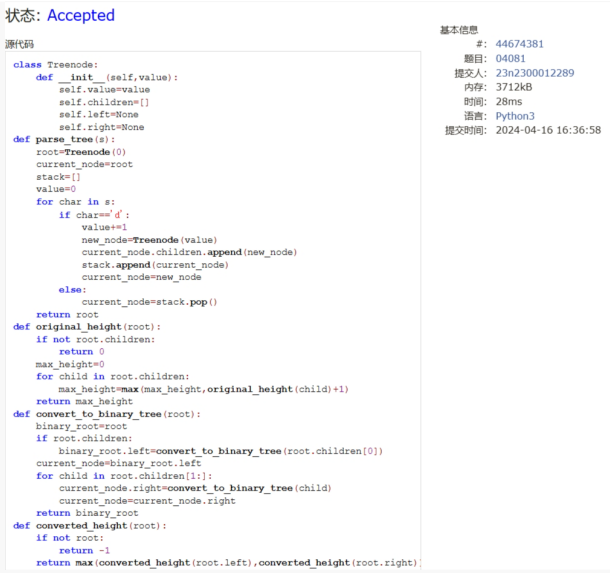
http://cs101.openjudge.cn/dsapre/04081/

思路：根据输入建普通树→求普通树的高度→将普通树转化为二叉树→求二叉树的高度

代码

```
class Treenode:
    def __init__(self,value):
        self.value=value
        self.children=[]
        self.left=None
        self.right=None
def parse_tree(s):
    root=Treenode(0)
    current_node=root
    stack=[]
    value=0
    for char in s:
        if char=='d':
            value+=1
            new_node=Treenode(value)
            current_node.children.append(new_node)
            stack.append(current_node)
            current_node=new_node
        else:
            current_node=stack.pop()
    return root
def original_height(root):
    if not root.children:
        return 0
    max_height=0
    for child in root.children:
        max_height=max(max_height,original_height(child)+1)
    return max_height
def convert_to_binary_tree(root):
    binary_root=root
    if root.children:
        binary_root.left=convert_to_binary_tree(root.children[0])
    current_node=binary_root.left
    for child in root.children[1:]:
        current_node.right=convert_to_binary_tree(child)
        current_node=current_node.right
    return binary_root
def converted_height(root):
    if not root:
        return -1
    return max(converted_height(root.left),converted_height(root.right))+1
s=input()
root=parse_tree(s)
original=original_height(root)
binary_root=convert_to_binary_tree(root)
converted=converted_height(binary_root)
print(f"{original} => {converted}")
```

代码运行截图（至少包含有"Accepted"）




## 08581: 扩展二叉树

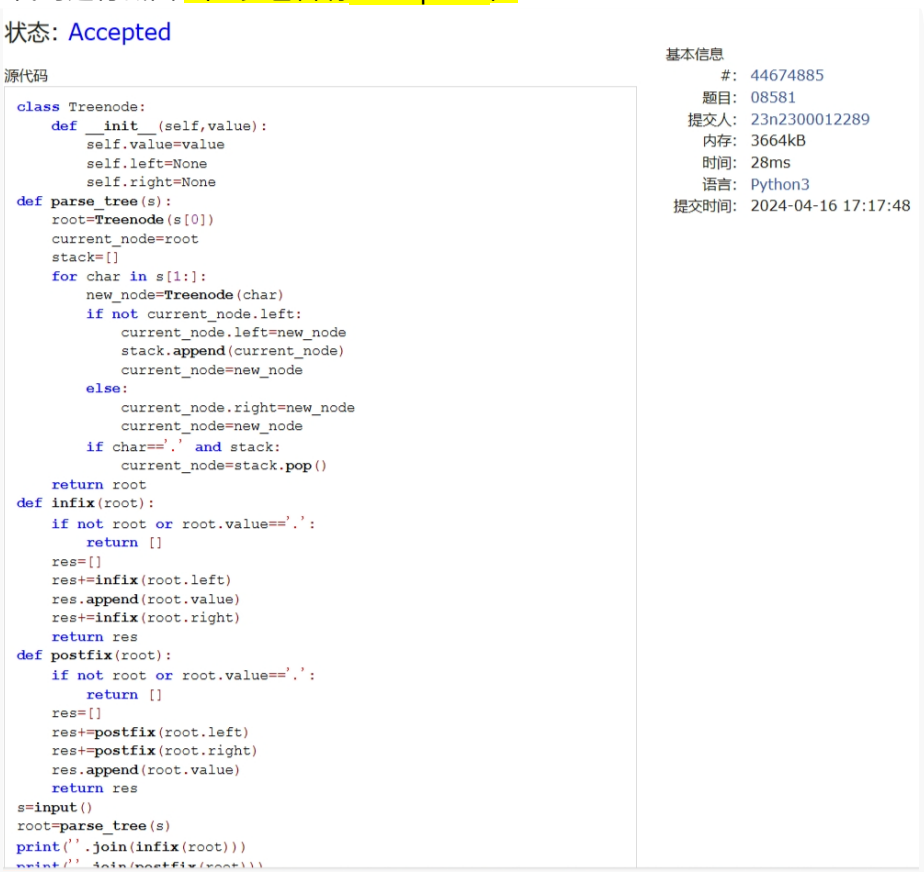
http://cs101.openjudge.cn/dsapre/08581/

思路：用栈存还没有右孩子的节点，遇到'.'就弹出栈顶节点，建树后根据定义遍历得到中/后序表达式

代码

```
class Treenode:
    def __init__(self,value):
        self.value=value
        self.left=None
        self.right=None
def parse_tree(s):
    root=Treenode(s[0])
    current_node=root
    stack=[]
    for char in s[1:]:
        new_node=Treenode(char)
        if not current_node.left:
            current_node.left=new_node
            stack.append(current_node)
            current_node=new_node
        else:
            current_node.right=new_node
            current_node=new_node
        if char=='.' and stack:
            current_node=stack.pop()
    return root
def infix(root):
    if not root or root.value=='.':
        return []
    res=[]
    res+=infix(root.left)
    res.append(root.value)
    res+=infix(root.right)
    return res
def postfix(root):
    if not root or root.value=='.':
        return []
    res=[]
    res+=postfix(root.left)
    res+=postfix(root.right)
    res.append(root.value)
    return res
s=input()
root=parse_tree(s)
print(''.join(infix(root)))
print(''.join(postfix(root)))
```

代码运行截图 （至少包含有"Accepted"）




## 22067: 快速堆猪

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

思路：最开始想尝试一个栈完成上述所有操作，毫不意外的min()超时；然后想用堆得到最小的重量，结果依然超时；看了题解发现开了两个栈，立马get到了意思然后成功AC

代码

```
stack=[]
min_stack=[]
while True:
    try:
        command=input().split()
        if command[0]=='push':
            pig=int(command[1])
            stack.append(pig)
            if not min_stack:
                min_stack.append(pig)
            else:
                min_stack.append(min(pig,min_stack[-1]))
        elif command[0]=='pop':
            if stack:
                stack.pop()
                min_stack.pop()
        else:
            if min_stack:
                print(min_stack[-1])
    except EOFError:
        break
```

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&回溯，注意遍历完所有移动方向/已走完棋盘所有点时将该点恢复为未访问的状态

代码

```
def dfs(n,m,x,y,visited,count):
    ans=0
    if x<0 or x>n-1 or y<0 or y>m-1 or visited[x][y]:
        return 0
    visited[x][y]=True
    if count==n*m-1:
        visited[x][y]=False
        return 1
    for (dx,dy) in [(1,2),(2,1),(-1,2),(2,-1),(1,-2),(-2,1),(-1,-2),(-2,-1)]:
        ans+=dfs(n,m,x+dx,y+dy,visited,count+1)
    visited[x][y]=False
    return ans
T=int(input())
for _ in range(T):
    n,m,x,y=map(int,input().split())
    visited=[[False]*m for i in range(n)]
    ans=dfs(n,m,x,y,visited,0)
    print(ans)
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）


## 28046: 词梯

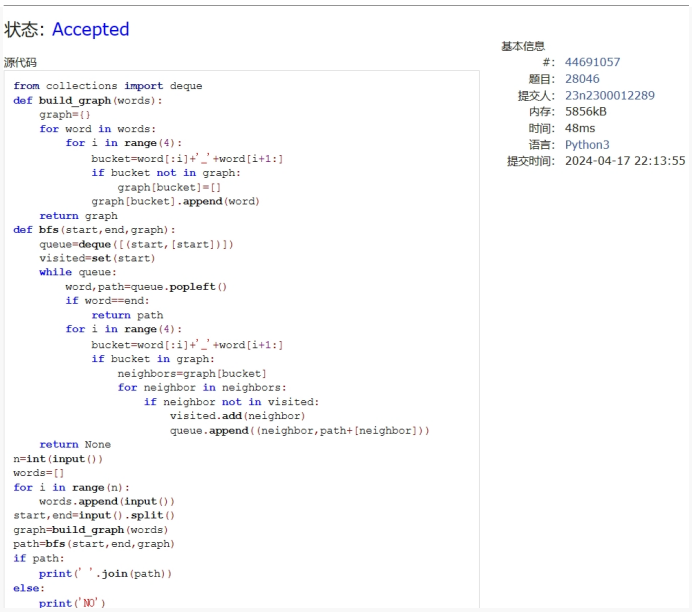
bfs, http://cs101.openjudge.cn/practice/28046/

思路：如果直接两两比较，时间复杂度是o（n^2），大概率会超时；在B站上找到了陈斌老师对这道题目的详细讲解，知道了可以用桶优化，然后bfs即可，不过我对这个代码有个疑问：如何保证输出的一定是最短路径？

代码

```
from collections import deque
def build_graph(words):
    graph={}
    for word in words:
        for i in range(4):
            bucket=word[:i]+'_'+word[i+1:]
            if bucket not in graph:
                graph[bucket]=[]
            graph[bucket].append(word)
    return graph
def bfs(start,end,graph):
    queue=deque([(start,[start])])
    visited=set(start)
    while queue:
        word,path=queue.popleft()
        if word==end:
            return path
        for i in range(4):
            bucket=word[:i]+'_'+word[i+1:]
            if bucket in graph:
                neighbors=graph[bucket]
                for neighbor in neighbors:
                    if neighbor not in visited:
                        visited.add(neighbor)
                        queue.append((neighbor,path+
[neighbor]))    return None
n=int(input())
words=[]
for i in range(n):
    words.append(input())
start,end=input().split()
graph=build_graph(words)
path=bfs(start,end,graph)
if path:
    print(' '.join(path))
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：原以为和马走日一样，结果超时严重。然后通过学习知道可以用Warnsdorff's rule进行优化，每次移动选择有最少未访问邻居的格子，但是没太明白这样做为什么可以降低时间复杂度

代码

```
def knight_tour(n,sr,sc):
    visited=[[False]*n for _ in range(n)]
    moves=[(2,1),(1,2),(-1,2),(-2,1),(-2,-1),(-1,-2),(1,-2),(2,-1)]
    def is_valid_move(r,c):
        return 0<=r<n and 0<=c<n and not visited[r][c]
    def count_valid_move(r,c):
        count=0
        for dr,dc in moves:
            nr,nc=dr+r,dc+c
            if is_valid_move(nr,nc):
                count+=1
        return count
    def get_next_move(r,c):
        min_count=9
        next_move=None
        for dr,dc in moves:
            nr,nc=dr+r,dc+c
            if is_valid_move(nr,nc):
                count=count_valid_move(nr,nc)
                if count<min_count:
                    min_count=count
                    next_move=(nr,nc)
        return next_move
    def dfs(r,c,count):
        visited[r][c]=True
        count+=1
        if count==n*n:
            return True
        next_move=get_next_move(r,c)
        if next_move:
            nr,nc=next_move
            if dfs(nr,nc,count):
                return True
        visited[r][c]=False
        return False
    return 'success' if dfs(sr,sc,0) else 'fail'
n=int(input())
sr,sc=map(int,input().split())
print(knight_tour(n,sr,sc))
```

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。

从词梯和骑士周游学到了很多！从上学期就看见群里的大佬们一直在说桶这个数据结构，今天总算是见识到了它的巧妙之处，接下来会找时间学习一下；Warnsdorff's rule对dfs进行优化，可以提高查找效率；关于回溯，千万不要忘记将节点恢复未访问的状态

不过这次作业还是遇到了两个问题：词梯的代码虽然AC，但是没太搞明白是如何保证输出的是最短路径；Warnsdorff's rule中为什么优先选择能够到达最少未访问的格子的位置作为下一步移动的位置可以加速搜索过程