Assignment #9: 图论：遍历，及 树算

Updated 1739 GMT+8 Apr 14, 2024

2024 spring, Complied by 赵策 数学科学学院

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

操作系统：Windows 11

Python编程环境：Visual Studio Code 1.86.2

1. 题目

04081: 树的转换

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

思路：

代码

```
#
def tree_height1(s):
    height=0
    max_height=0
    for char in s:
        if char=='d':
            height+=1
            max_height=max(max_height,height)
        else:
            height-=1
    return max_height

def tree_height2(s):
    height=0
    stack=[]
```

```
    max_height=0
    for char in s:
        if char=='d':
            height+=1
            stack.append(height)
            max_height=max(max_height,height)
        else:
            height=stack.pop()
    return max_height

s=input()
print(f'{tree_height1(s)} => {tree_height2(s)}')
```

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：

代码

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

def build_tree(preorder):
    if not preorder:
        return None
    root_val=preorder.pop(0)
    if root_val=='.':
        return None
    root=TreeNode(root_val)
    root.left=build_tree(preorder)
    root.right=build_tree(preorder)
    return root

def inorder_traversal(root):
    if not root:
        return []
    return inorder_traversal(root.left)+[root.val]+inorder_traversal(root.right)

def postorder_traversal(root):
```

```
    if not root:
        return []
    return postorder_traversal(root.left)+postorder_traversal(root.right)+
[root.val]

preorder=list(input())
root=build_tree(preorder)
print(''.join(inorder_traversal(root)))
print(''.join(postorder_traversal(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：

代码

```
#
class PigStack:
    def __init__(self):
        self.pig_stack=[]
        self.min_weights=[]

    def push(self,weight):
        self.pig_stack.append(weight)
        if not self.min_weights or weight<self.min_weights[-1]:
            self.min_weights.append(weight)
        else:
            self.min_weights.append(self.min_weights[-1])

    def pop(self):
        if self.pig_stack:
            self.pig_stack.pop()
            self.min_weights.pop()

    def get_min_weight(self):
        if not self.min_weights:
            return None
        return self.min_weights[-1]

pig_stack=PigStack()
while True:
    try:
        command=input().split()
```

```
        if command[0]=='push':
            pig_stack.push(int(command[1]))
        elif command[0]=='pop':
            pig_stack.pop()
        else:
            min_weight=pig_stack.get_min_weight()
            if min_weight is not None:
                print(min_weight)
    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：

代码

```
#
def is_valid_move(x,y,n,m):
    if x<0 or y<0 or x>=n or y>=m or visited[x][y]:
        return False
    return True

def dfs(x,y,n,m,count):
    if count==n*m:
        return 1
    total_paths=0
    dx=[-2,-1,1,2,2,1,-1,-2]
    dy=[1,2,2,1,-1,-2,-2,-1]
    for i in range(8):
        new_x=x+dx[i]
        new_y=y+dy[i]
        if is_valid_move(new_x,new_y,n,m):
            visited[new_x][new_y]=True
            total_paths+=dfs(new_x,new_y,n,m,count+1)
            visited[new_x][new_y]=False
    return total_paths

for _ in range(int(input())):
    n,m,x,y=map(int,input().split())
    visited=[[False]*m for _ in range(n)]
    visited[x][y]=True
```

```
    print(dfs(x,y,n,m,1))
```

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：

代码

```
#
from collections import defaultdict,deque
def categorize(word_list):
    categorization=defaultdict(list)
    for word in word_list:
        for i in range(4):
            pattern=word[:i]+'*'+word[i+1:]
            categorization[pattern].append(word)
    return categorization

def bfs(start_word,end_word):
    queue=deque([(start_word,[start_word])])
    visited=set([start_word])
    while queue:
        current_word,path=queue.popleft()
        if current_word==end_word:
            return path
        for i in range(4):
            pattern=current_word[:i]+'*'+current_word[i+1:]
            for neighbor in categorization[pattern]:
                if neighbor not in visited:
                    visited.add(neighbor)
                    queue.append((neighbor,path+[neighbor]))
    return []

n=int(input())
word_list=[input() for _ in range(n)]
start_word,end_word=input().split()
categorization=categorize(word_list)
shortest_path=bfs(start_word,end_word)
if shortest_path:
    print(' '.join(shortest_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：

代码

```
#
def is_valid(x,y):
    return 0<=x<n and 0<=y<n and not visited[x][y]

def sort_neighbors(x,y):
    directions=[(-2,-1),(-2,1),(-1,-2),(-1,2),(1,-2),(1,2),(2,-1),(2,1)]
    neighbors=[]
    for dx,dy in directions:
        nx,ny=x+dx,y+dy
        if is_valid(nx,ny):
            count=0
            for dx1,dy1 in directions:
                if is_valid(nx+dx1,ny+dy1):
                    count+=1
            neighbors.append((nx,ny,count))
    return sorted(neighbors,key=lambda x:x[2])

def knight_tour(x,y,count):
    visited[x][y]=True
    count+=1
    if count==n*n:
        return True
    for nx,ny,_ in sort_neighbors(x,y):
        if knight_tour(nx,ny,count):
            return True
    visited[x][y]=False
    return False

n=int(input())
visited=[[False]*n for _ in range(n)]
x,y=map(int,input().split())
print('success' if knight_tour(x,y,0) else 'fail')
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

28050: 骑士周游 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。

```
#词梯直接建图会TLE，而优化之后通过时间大概在49ms，两种写法差距巨大(后面bfs差不多，最开始优化还想过bidirectional_bfs)；主要就是类似通配符的想法分类
#Ver1
def build_graph(word_list):
    graph={}
    for word in word_list:
        graph[word]=[]
        for other_word in word_list:
            if is_neighbor(word,other_word):
                graph[word].append(other_word)
    return graph

def is_neighbor(word1,word2):
    diff_count=0
    for i in range(4):
        if word1[i]!=word2[i]:
            diff_count+=1
    return diff_count==1
#Ver2
def categorize(word_list):
    categorization=defaultdict(list)
    for word in word_list:
        for i in range(4):
            pattern=word[:i]+'*'+word[i+1:]
            categorization[pattern].append(word)
    return categorization

#骑士周游还是得用warnsdorff：沃恩斯多夫规则是一种启发式算法，用于在每一步选择下一个可能移动的位置。该规则指导我们选择下一个移动的位置是那个在未访问的邻居中具有最少未访问邻居的位置。这样做可以大大减少搜索空间。
```