Assignment #9: 图论：遍历，及 树算

Updated 1739 GMT+8 Apr 14, 2024

2024 spring, Complied by 胡登科、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：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

04081: 树的转换

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

思路：

```
# 先定义树高函数 便是记录老序列 d能d到的最下面的点 然后每次u的时候现在的记录点就低一点
# 新序列的记录方法 先d如果遇到最下面的分叉时 会少d一下 但是我们的记录任然存着 会在下一次d的时候回来 所以可以记录到新的d
```

代码

```
#
def tree_height(s):
    old_height = 0
    max_old = 0
    new_height = 0
    max_new = 0
    stack = []
    for c in s:
```

```
        if c == "d":
            old_height += 1
            max_old = max(max_old, old_height)

            new_height += 1
            stack.append(new_height)
            max_new = max(max_new, new_height)
        else:
            old_height -= 1
            new_height = stack[-1]
            stack.pop()
    return f"{max_old} => {max_new}"


s = input().strip()
print(tree_height(s))
```

代码运行截图 （至少包含有"Accepted"）

状态：Accepted

源代码

```
# 树的转换
# 先定义树高函数 便是记录老序列 d能d到的最下面的点 然后每次u的时候现在的记录点就低一
# 新序列的记录方法  先d如果遇到最下面的分叉时 会少d一下 但是我们的记录任然存着 会在
def tree_height(s):
    old_height = 0
    max_old = 0
    new_height = 0
    max_new = 0
    stack = []
    for c in s:
        if c == "d":
            old_height += 1
            max_old = max(max_old, old_height)

            new_height += 1
            stack.append(new_height)
            max_new = max(max_new, new_height)
        else:
            old_height -= 1
            new_height = stack[-1]
            stack.pop()
    return f"{max_old} => {max_new}"


s = input().strip()
print(tree_height(s))
```

08581: 扩展二叉树

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

思路:

```
# 先通过前序序列遍历建树 是一道典型的前中后序的题目
# 难点在于建树的过程 利用.建立左右均为None的结构
```

代码

```
#
def build_tree(preorder):
    if not preorder or preorder[0] == '.':
        return None, preorder[1:]
    root = preorder[0]
    left, preorder = build_tree(preorder[1:])
    right, preorder = build_tree(preorder)
    return (root, left, right), preorder


def inorder(tree):
    if tree is None:
        return ''
    root, left, right = tree
    return inorder(left) + root + inorder(right)


def postorder(tree):
    if tree is None:
        return ''
    root, left, right = tree
    return postorder(left) + postorder(right) + root


preorder = input().strip()

tree, _ = build_tree(preorder)
print(inorder(tree))
print(postorder(tree))
```

代码运行截图 （至少包含有"Accepted"）

#44765508提交状态 查看 提交

状态: Accepted

源代码

```
def build_tree(preorder):
    if not preorder or preorder[0] == '.':
        return None, preorder[1:]
    root = preorder[0]
    left, preorder = build_tree(preorder[1:])
    right, preorder = build_tree(preorder)
    return (root, left, right), preorder


def inorder(tree):
    if tree is None:
        return ''
    root, left, right = tree
    return inorder(left) + root + inorder(right)


def postorder(tree):
    if tree is None:
        return ''
    root, left, right = tree
    return postorder(left) + postorder(right) + root


preorder = input().strip()

tree, _ = build_tree(preorder)
print(inorder(tree))
print(postorder(tree))
```

基本信息

#: 44765508
题目: 08581
提交人: 22000122
内存: 5816kB
时间: 30ms
语言: Python3
提交时间: 2024-04-

22067: 快速堆猪

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

思路：简单的栈的问题 巧妙之处在于把所有的参数的记录成当前的最小值 输出就不用再检索栈中最小元素

代码

```
#
pig_stack = []
m = []
while True:
    try:
        s = input().split()

        if s[0] == 'pop':
            if pig_stack:
                pig_stack.pop()
                if m:
                    m.pop()
        elif s[0] == 'min':
```

```
            if m:
                print(m[-1])
        else:
            h = int(s[1])
            pig_stack.append(h)
            if not m:
                m.append(h)
            else:
                k = m[-1]
                m.append(min(k,h))


    except EOFError:
        break
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

状态：Accepted

源代码

```
# 快速堆猪
# 栈的典型题目
pig_stack = []
m = []
while True:
    try:
        s = input().split()

        if s[0] == 'pop':
            if pig_stack:
                pig_stack.pop()
                if m:
                    m.pop()
        elif s[0] == 'min':
            if m:
                print(m[-1])
        else:
            h = int(s[1])
            pig_stack.append(h)
            if not m:
                m.append(h)
            else:
                k = m[-1]
                m.append(min(k,h))


    except EOFError:
        break
```

04123: 马走日

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

思路：

看完答案也没明白了。

代码

```
# s = int(input())
for _ in range(s):
    m, n, a, b = map(int, input().split())
    dirs = [(2, 1), (1, 2), (-2, 1), (-1, 2), (2, -1), (1, -2), (-1, -2), (-2,
-1)]
    borad = [[0] * n for x in range(m)]
    cut = 0


    def move(x, y, step):
        global cut
        if step == n * m:
            cut += 1
            return
        for i in range(8):
            move_x = x + dirs[i][0]  # 表示列表中第i个元组的第一个元素
            move_y = y + dirs[i][1]
            if move_x >= 0 and move_x <= m - 1 and move_y >= 0 and move_y <= n -
1 and borad[move_x][move_y] == 0:
                borad[move_x][move_y] = 1
                move(move_x, move_y, step + 1)
                borad[move_x][move_y] = 0


    borad[a][b] = 1
    move(a, b, 1)
    print(cut)
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

状态: Accepted

源代码

```
s = int(input())
for _ in range(s):
    m, n, a, b = map(int, input().split())
    dirs = [(2, 1), (1, 2), (-2, 1), (-1, 2), (2, -1), (1, -2), (-1, -2)
    borad = [[0] * n for x in range(m)]
    cut = 0


    def move(x, y, step):
        global cut
        if step == n * m:
            cut += 1
            return
        for i in range(8):
            move_x = x + dirs[i][0]  # 表示列表中第i个元组的第一个元素
            move_y = y + dirs[i][1]
            if move_x >= 0 and move_x <= m - 1 and move_y >= 0 and move_
                borad[move_x][move_y] = 1
                move(move_x, move_y, step + 1)
                borad[move_x][move_y] = 0


    borad[a][b] = 1
    move(a, b, 1)
    print(cut)
```

28046: 词梯

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

思路:

不是很明白怎么做这个题目。

代码

```
#
import sys
from collections import deque

class Graph:
    def __init__(self):
        self.vertices = {}
        self.num_vertices = 0

    def add_vertex(self, key):
        self.num_vertices = self.num_vertices + 1
        new_vertex = Vertex(key)
        self.vertices[key] = new_vertex
        return new_vertex

    def get_vertex(self, n):
        if n in self.vertices:
            return self.vertices[n]
        else:
```

```
            return None

    def __len__(self):
        return self.num_vertices

    def __contains__(self, n):
        return n in self.vertices

    def add_edge(self, f, t, cost=0):
        if f not in self.vertices:
            nv = self.add_vertex(f)
        if t not in self.vertices:
            nv = self.add_vertex(t)
        self.vertices[f].add_neighbor(self.vertices[t], cost)

    def get_vertices(self):
        return list(self.vertices.keys())

    def __iter__(self):
        return iter(self.vertices.values())


class Vertex:
    def __init__(self, num):
        self.key = num
        self.connectedTo = {}
        self.color = 'white'
        self.distance = sys.maxsize
        self.previous = None
        self.disc = 0
        self.fin = 0

    def add_neighbor(self, nbr, weight=0):
        self.connectedTo[nbr] = weight

    # def setDiscovery(self, dtime):
    #     self.disc = dtime
    #
    # def setFinish(self, ftime):
    #     self.fin = ftime
    #
    # def getFinish(self):
    #     return self.fin
    #
    # def getDiscovery(self):
    #     return self.disc

    def get_neighbors(self):
        return self.connectedTo.keys()

    # def getWeight(self, nbr):
    #     return self.connectedTo[nbr]

    # def __str__(self):
    #     return str(self.key) + ":color " + self.color + ":disc " +
str(self.disc) + ":fin " + str(
```

```
    #         self.fin) + ":dist " + str(self.distance) + ":pred \n\t[" +
str(self.previous) + "]\n"



def build_graph(all_words):
    buckets = {}
    the_graph = Graph()

    # 创建词桶 create buckets of words that differ by 1 letter
    for line in all_words:
        word = line.strip()
        for i, _ in enumerate(word):
            bucket = f"{word[:i]}_{word[i + 1:]}"
            buckets.setdefault(bucket, set()).add(word)

    # 为同一个桶中的单词添加顶点和边
    for similar_words in buckets.values():
        for word1 in similar_words:
            for word2 in similar_words - {word1}:
                the_graph.add_edge(word1, word2)

    return the_graph


def bfs(start, end):
    start.distnce = 0
    start.previous = None
    vert_queue = deque()
    vert_queue.append(start)
    while len(vert_queue) > 0:
        current = vert_queue.popleft()  # 取队首作为当前顶点

        if current == end:
            return True

        for neighbor in current.get_neighbors():  # 遍历当前顶点的邻接顶点
            if neighbor.color == "white":
                neighbor.color = "gray"
                neighbor.distance = current.distance + 1
                neighbor.previous = current
                vert_queue.append(neighbor)
        current.color = "black"  # 当前顶点已经处理完毕，设黑色

    return False

"""
BFS 算法主体是两个循环的嵌套: while-for
    while 循环对图中每个顶点访问一次，所以是 O(|V|)；
    嵌套在 while 中的 for，由于每条边只有在其起始顶点u出队的时候才会被检查一次，
    而每个顶点最多出队1次，所以边最多被检查次，一共是 O(|E|)；
    综合起来 BFS 的时间复杂度为 O(V+|E|)

词梯问题还包括两个部分算法
    建立 BFS 树之后，回溯顶点到起始顶点的过程，最多为 O(|V|)
    创建单词关系图也需要时间，时间是 O(|V|+|E|) 的，因为每个顶点和边都只被处理一次
```

```
"""


def traverse(starting_vertex):
    ans = []
    current = starting_vertex
    while (current.previous):
        ans.append(current.key)
        current = current.previous
    ans.append(current.key)

    return ans


n = int(input())
all_words = []
for _ in range(n):
    all_words.append(input().strip())

g = build_graph(all_words)
# print(len(g))

s, e = input().split()
start, end = g.get_vertex(s), g.get_vertex(e)
if start is None or end is None:
    print('NO')
    exit(0)

if 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"）

状态: Accepted

源代码

```
import sys
from collections import deque

class Graph:
    def __init__(self):
        self.vertices = {}
        self.num_vertices = 0

    def add_vertex(self, key):
        self.num_vertices = self.num_vertices + 1
        new_vertex = Vertex(key)
        self.vertices[key] = new_vertex
        return new_vertex

    def get_vertex(self, n):
        if n in self.vertices:
            return self.vertices[n]
        else:
            return None

    def __len__(self):
        return self.num_vertices

    def __contains__(self, n):
        return n in self.vertices

    def add_edge(self, f, t, cost=0):
        if f not in self.vertices:
            nv = self.add_vertex(f)
        if t not in self.vertices:
            nv = self.add_vertex(t)
        self.vertices[f].add_neighbor(self.vertices[t], cost)
```

28050: 骑士周游

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

思路：不是很会做 参考了同学的代码

代码

```
#
def knight_tour(n, sr, sc):
    moves = [(-2, -1), (-2, 1), (-1, -2), (-1, 2),
             (1, -2), (1, 2), (2, -1), (2, 1)]

    visited = [[False] * n for _ in range(n)]

    def is_valid_move(row, col):
```

```
        return 0 <= row < n and 0 <= col < n and not visited[row][col]

    def count_neighbors(row, col):
        count = 0
        for dr, dc in moves:
            next_row, next_col = row + dr, col + dc
            if is_valid_move(next_row, next_col):
                count += 1
        return count

    def sort_moves(row, col):
        neighbor_counts = []
        for dr, dc in moves:
            next_row, next_col = row + dr, col + dc
            if is_valid_move(next_row, next_col):
                count = count_neighbors(next_row, next_col)
                neighbor_counts.append((count, (next_row, next_col)))
        neighbor_counts.sort()
        sorted_moves = [move[1] for move in neighbor_counts]
        return sorted_moves

    visited[sr][sc] = True
    tour = [(sr, sc)]

    while len(tour) < n * n:
        current_row, current_col = tour[-1]
        sorted_next_moves = sort_moves(current_row, current_col)
        if not sorted_next_moves:
            return "fail"
        next_row, next_col = sorted_next_moves[0]
        visited[next_row][next_col] = True
        tour.append((next_row, next_col))

    return "success"


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"）

状态: Accepted

源代码

```
def knight_tour(n, sr, sc):
    moves = [(-2, -1), (-2, 1), (-1, -2), (-1, 2),
             (1, -2), (1, 2), (2, -1), (2, 1)]

    visited = [[False] * n for _ in range(n)]

    def is_valid_move(row, col):
        return 0 <= row < n and 0 <= col < n and not visited[row][col]

    def count_neighbors(row, col):
        count = 0
        for dr, dc in moves:
            next_row, next_col = row + dr, col + dc
            if is_valid_move(next_row, next_col):
                count += 1
        return count

    def sort_moves(row, col):
        neighbor_counts = []
        for dr, dc in moves:
            next_row, next_col = row + dr, col + dc
            if is_valid_move(next_row, next_col):
                count = count_neighbors(next_row, next_col)
                neighbor_counts.append((count, (next_row, next_col)))
        neighbor_counts.sort()
        sorted_moves = [move[1] for move in neighbor_counts]
        return sorted_moves

    visited[sr][sc] = True
```

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。

这周的题目前三道简单，后三道难。感觉才把递归学会现在有需要学习深搜和广搜了，真是不简单那。我看看陈斌老师的网课感觉讲的挺明白的。加油