Assignment #9: 图论：遍历，及 树算

Updated 1739 GMT+8 Apr 14, 2024

2024 spring, Complied by 赵语涵 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

操作系统：windows 11

Python编程环境：Spyder IDE 5.2.2

1. 题目

04081: 树的转换

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

思路：之前感觉不会做树的转换所以把这道题空了一段时间，返回来再看发现其实很好做。用栈储存还没有右节点的层数，输入为'u'时跳到pop出的无右节点的层数即可。

代码

```
#赵语涵2300012254
x = list(input())
l = len(x)

h_1 = 0
now = 0
for i in x:
    if i == 'd':
        now += 1
    else:
        now -= 1
    h_1 = max(h_1,now)

h_2 = 0
```

```
now = 0
right = [0,]
for i in x:
    if i == 'd':
        now += 1
        right.append(now)
    else:
        now = right.pop()
    h_2 = max(h_2,now)

print(f'{h_1} => {h_2}')
```

代码运行截图

#45194935提交状态

查看 提交 统计

状态: Accepted

源代码

```
#赵语涵2300012254
x = list(input())
l = len(x)

h_1 = 0
now = 0
```

基本信息

#: 45194935
题目: 04081
提交人: 23n2300012254
内存: 3652kB
时间: 20ms
语言: Python3
提交时间: 2024-06-04 10:56:09

08581: 扩展二叉树

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

思路：1）建树，从输入开头依次弹出字符建立为当前节点的左子树，且弹出为'.'时表示为末节点，到达末节点后一直向上回溯直到可以填入空的右子树为止

2）中、后序输出，递归处理，只是对于左、右、当前节点处理的顺序不同。当左右子树值均为'.'时停止递归返回上一级

代码

```
#赵语涵2300012254
class Node():
    def __init__(self,value,parent):
        self.value = value
        if self.value == '.':  #当值为'.'时表示到达树的末端，设置左右子树填满
            self.left = 1
            self.right = 1
        else:
            self.left = ''
            self.right = ''
        self.parent = parent

```

```python
pre = list(input())[::-1]
present = Node('','')
start = present
start.right = Node('.',start)
while pre:   #建立树
    i = pre.pop()
    if not present.left:
        present.left = (n:=Node(i,present))
    elif not present.right:
        present.right = (n:=Node(i,present))
    else:
        present = present.parent
        while present.right:
            present = present.parent
        present.right = (n:=Node(i,present))
    present = n

mid,post = '',''
def write_post(node):   #后序序列
    global post
    if node.left.value != '.':
        write_post(node.left)
    if node.right.value != '.':
        write_post(node.right)
    post += node.value


def write_mid(node):
    global mid
    if node.left.value != '.':
        write_mid(node.left)
    mid += node.value
    if node.right.value != '.':
        write_mid(node.right)

write_mid(start)
print(mid)
write_post(start)
print(post)
```

代码运行截图

#44878065提交状态

查看 提交 统计

状态: Accepted

源代码

```python
#赵语涵2300012254
class Node():
    def __init__(self,value,parent):
        self.value = value
        if self.value == '.':  #当值为'.'时表示到达树的末端，设置左右子树填满
            self.left = 1
```

基本信息

#: 44878065
题目: 08581
提交人: 23n2300012254
内存: 3668kB
时间: 23ms
语言: Python3
提交时间: 2024-05-06 10:47:37

22067: 快速堆猪

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

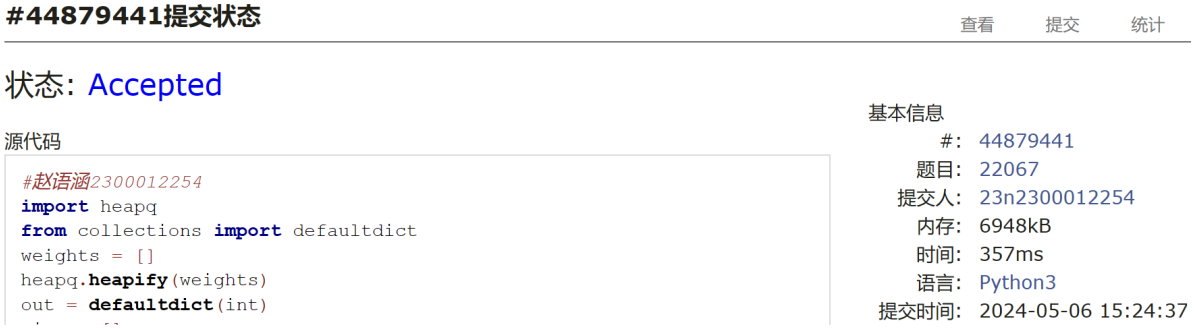
思路：虽然想到了用heapq，但是尝试把放入顺序同时存在heapq中，反而感觉写起来很麻烦。看题解，多存几个参数，用weights（heapq）排序存储重量，pigs（list）存储实际push和pop的猪，注意有out（dict），用于存储某个重量的猪的数量，这样在heapq弹出最小猪的时候如果实际这个重量的猪已经没有了就会继续弹出下一只猪

代码

```
#赵语涵2300012254
import heapq
from collections import defaultdict
weights = []
heapq.heapify(weights)
out = defaultdict(int)
pigs = []
while True:
    try:
        ope = input()
        if ope== 'pop':
            if pigs:
                out[pigs.pop()] -= 1
        elif ope[:2]=='pu':
            pigs.append((n:=int(ope.split()[1])))
            out[n] += 1
            heapq.heappush(weights, n)
        else:
            if pigs:
                while out[(a:=heapq.heappop(weights))] <= 0:
                        continue
                print(a)
                heapq.heappush(weights, a)
    except EOFError:
        break#

```

代码运行截图



#44879441提交状态

查看　提交　统计

状态: Accepted

源代码

```
#赵语涵2300012254
import heapq
from collections import defaultdict
weights = []
heapq.heapify(weights)
out = defaultdict(int)
```

基本信息

#: 44879441
题目: 22067
提交人: 23n2300012254
内存: 6948kB
时间: 357ms
语言: Python3
提交时间: 2024-05-06 15:24:37

04123: 马走日

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

思路：很容易想到dfs，需要考虑的是回溯和找到途径的条件。找到途径比较容易考虑，加一个经过点数目的参数count，当count==n*m时即完成遍历所有点。对我比较难的是回溯条件。最开始在for change结束后使mark[x][y]=True，然而结果错误，参照题解应当在每次dfs后还原当前位点。

代码

```python
#赵语涵2300012254
num = 0
def dfs(x,y,count):
    global num
    mark[x][y] = False
    if count == n*m:
        num += 1
        return
    for change in ways:
        a,b = change[0],change[1]
        if 0<=x+a<n and 0<=y+b<m:
            if mark[x+a][y+b]:
                dfs(x+a,y+b,count+1)
                mark[x+a][y+b]=True

ways = [(1,2),(2,1),(2,-1),(1,-2),(-1,-2),(-2,-1),(-2,1),(-1,2)]

for _ in range(int(input())):
    n,m,x,y = map(int,input().split())
    mark = {}
    for a in range(n):
        mark[a] = {}
        for b in range(m):
            mark[a][b] = True
    num = 0
    dfs(x,y,1)
    print(num)
```

代码运行截图

#44901954提交状态

查看　提交　统计

状态: Accepted

源代码

```
#赵语涵2300012254
num = 0
def dfs(x,y,count):
    global num
    mark[x][y] = False
    if count == n*m:
        num += 1
```

基本信息

#: 44901954
题目: 04123
提交人: 23n2300012254
内存: 3616kB
时间: 3416ms
语言: Python3
提交时间: 2024-05-08 19:30:16

28046: 词梯

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

思路：很明显使用bfs的方法，关键在于数据结构的处理。最开始考虑建图的时候，直接用{node:[所有的邻节点]}的方式，然后bfs每一层存入新的路径，结果MLE了。根据题解提示，首先建图采用桶的方式，另外不再存储可能路径，而只存储每个节点的最佳父节点。

代码

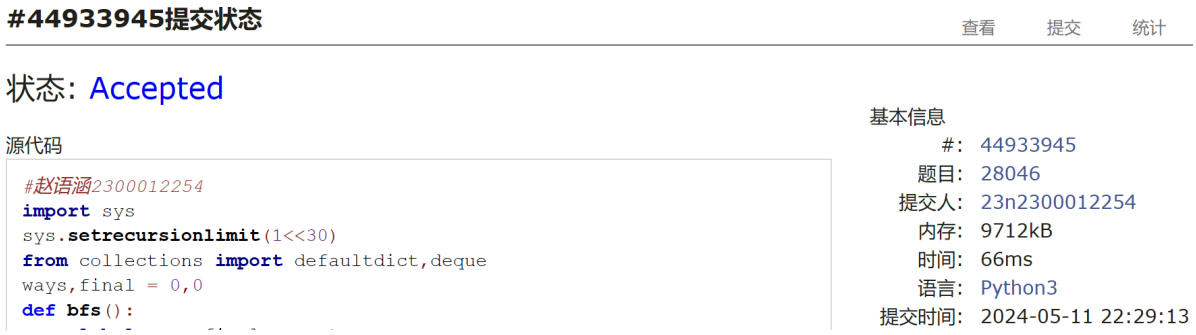
```
#赵语涵2300012254
import sys
sys.setrecursionlimit(1<<30)
from collections import defaultdict,deque
ways,final = 0,0
def bfs():
    global ways,final,parent
    while ways:
        word = ways.popleft()
        mark[word] = False
        for i in edges[word]:
            if mark[i]:
                parent[i] = word
                mark[i] = False
                ways.append(i)
                if final == i:
                    return True
    return False

def four(word):
    about = []
    for i in range(4):
        about.append(word[:i]+'_'+word[i+1:])
    return about

edges,mark = defaultdict(set),{}
barrel = defaultdict(list)
for _ in range(int(input())):
    for i in four((word:=input())):
        barrel[i].append(word)
    mark[word] = True
for i in barrel.values():
    for a in i:
        for b in i:
            edges[a].add(b)
            edges[b].add(a)
start,final = input().split()
ways = deque([start])
parent = {start:None}
```

```
mark[start] = False
if bfs():
    route = [(a:=final)]
    while (a:=parent[a]):
        route.append(a)
    print(' '.join(route[::-1]))
else:
    print('NO')
```

代码运行截图



28050: 骑士周游

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

思路：如果是dfs感觉还能做，看了题解感觉太复杂了，而且优化过程中考虑优先搜索下一个节点中下下个节点数目最少的，也是很难想到的部分。至于题解后面同学证明给出的不可能周游的规律感觉也不是非常友好。

代码

```
import sys

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

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

    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)
        #self.vertices[t].add_neighbor(self.vertices[f], cost)

    def getVertices(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 __lt__(self,o):
        return self.key < o.key

    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):
```

```
74      #     return self.connectedTo[nbr]
75
76      def __str__(self):
77          return str(self.key) + ":color " + self.color + ":disc " +
    str(self.disc) + ":fin " + str(
78              self.fin) + ":dist " + str(self.distance) + ":pred \n\t[" +
    str(self.previous) + "]\n"
79
80
81
82  def knight_graph(board_size):
83      kt_graph = Graph()
84      for row in range(board_size):           #遍历每一行
85          for col in range(board_size):       #遍历行上的每一个格子
86              node_id = pos_to_node_id(row, col, board_size) #把行、列号转为格子
    ID
87              new_positions = gen_legal_moves(row, col, board_size) #按照 马走
    日，返回下一步可能位置
88              for row2, col2 in new_positions:
89                  other_node_id = pos_to_node_id(row2, col2, board_size) #下一
    步的格子ID
90                  kt_graph.add_edge(node_id, other_node_id) #在骑士周游图中为两
    个格子加一条边
91      return kt_graph
92
93  def pos_to_node_id(x, y, bdSize):
94      return x * bdSize + y
95
96  def gen_legal_moves(row, col, board_size):
97      new_moves = []
98      move_offsets = [                        # 马走日的8种走法
99          (-1, -2),  # left-down-down
100         (-1, 2),  # left-up-up
101         (-2, -1),  # left-left-down
102         (-2, 1),  # left-left-up
103         (1, -2),  # right-down-down
104         (1, 2),  # right-up-up
105         (2, -1),  # right-right-down
106         (2, 1),  # right-right-up
107     ]
108     for r_off, c_off in move_offsets:
109         if (                                # #检查，不能走出棋盘
110             0 <= row + r_off < board_size
111             and 0 <= col + c_off < board_size
112         ):
113             new_moves.append((row + r_off, col + c_off))
114     return new_moves
115
116 # def legal_coord(row, col, board_size):
117 #     return 0 <= row < board_size and 0 <= col < board_size
118
119
120 def knight_tour(n, path, u, limit):
121     u.color = "gray"
122     path.append(u)              #当前顶点涂色并加入路径
123     if n < limit:
```

```
        neighbors = ordered_by_avail(u) #对所有的合法移动依次深入
        #neighbors = sorted(list(u.get_neighbors()))
        i = 0

        for nbr in neighbors:
            if nbr.color == "white" and \
                knight_tour(n + 1, path, nbr, limit):   #选择“白色”未经深入的点，层次加一，递归深入
                return True
        else:                           #所有的“下一步”都试了走不通
            path.pop()                  #回溯，从路径中删除当前顶点
            u.color = "white"           #当前顶点改回白色
            return False
    else:
        return True

def ordered_by_avail(n):
    res_list = []
    for v in n.get_neighbors():
        if v.color == "white":
            c = 0
            for w in v.get_neighbors():
                if w.color == "white":
                    c += 1
            res_list.append((c,v))
    res_list.sort(key = lambda x: x[0])
    return [y[1] for y in res_list]

# class DFSGraph(Graph):
#     def __init__(self):
#         super().__init__()
#         self.time = 0                   #不是物理世界，而是算法执行步数
#
#     def dfs(self):
#         for vertex in self:
#             vertex.color = "white"      #颜色初始化
#             vertex.previous = -1
#         for vertex in self:             #从每个顶点开始遍历
#             if vertex.color == "white":
#                 self.dfs_visit(vertex)  #第一次运行后还有未包括的顶点
#                                         # 则建立森林
#
#     def dfs_visit(self, start_vertex):
#         start_vertex.color = "gray"
#         self.time = self.time + 1       #记录算法的步骤
#         start_vertex.discovery_time = self.time
#         for next_vertex in start_vertex.get_neighbors():
#             if next_vertex.color == "white":
#                 next_vertex.previous = start_vertex
#                 self.dfs_visit(next_vertex)     #深度优先递归访问
#         start_vertex.color = "black"
#         self.time = self.time + 1
#         start_vertex.closing_time = self.time


def main():
```

```
    def NodeToPos(id):
       return ((id//8, id%8))

    bdSize = int(input())  # 棋盘大小
    *start_pos, = map(int, input().split())  # 起始位置
    g = knight_graph(bdSize)
    start_vertex = g.get_vertex(pos_to_node_id(start_pos[0], start_pos[1],
bdSize))
    if start_vertex is None:
        print("fail")
        exit(0)

    tour_path = []
    done = knight_tour(0, tour_path, start_vertex, bdSize * bdSize-1)
    if done:
        print("success")
    else:
        print("fail")

    exit(0)

    # 打印路径
    cnt = 0
    for vertex in tour_path:
        cnt += 1
        if cnt % bdSize == 0:
            print()
        else:
            print(vertex.key, end=" ")
            #print(NodeToPos(vertex.key), end=" ")   # 打印坐标

if __name__ == '__main__':
    main()
```

代码运行截图

#45194970提交状态

查看　提交　统计

状态: Accepted

源代码

```
import sys

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

基本信息

#: 45194970
题目: 28050
提交人: 23n2300012254
内存: 4100kB
时间: 25ms
语言: Python3
提交时间: 2024-06-04 11:00:18

2. 学习总结和收获

对树的练习加多了，并且也复习到了以前的dfs等写法。对于遇到的不会的题目，放一段时间也许会有思路（虽然考试不太适用、、