Assignment #A: 图论：遍历，树算及栈

Updated 2018 GMT+8 Apr 21, 2024

2024 spring, Complied by ~~天幂~~ 化学与分子工程学院

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 23H2

Python编程环境：Visual Studio Code 1.86.2230.

1. 题目

20743: 整人的提词本

http://cs101.openjudge.cn/practice/20743/

思路：通过模拟翻转的过程，遇到右括号通过挨个弹出直至遇到左括号来翻转；遇到其他符号直接入栈。额外套了两层括号避免处理边界case。

代码

```
class Node:
    def __init__(self, id):
        self.id = id
        self.children = []
        self.father = None

    def add_child(self, child):
        self.children.append(child)
        child.father = self
        return child

class Tree:
    def __init__(self):
        self.root = Node(0)
```

```
        self.nodes = {0: self.root}
        self.len = 1
        self.pointer = self.root

    def move(self, op):
        if op == 'd':
            self.pointer = self.pointer.add_child(Node(self.len))
            self.nodes[self.len] = self.pointer
            self.len += 1
        if op == 'u':
            self.pointer = self.pointer.father

    def depth(self):
        return self._depth(self.root)

    @staticmethod
    def _depth(node):
        if len(node.children) == 0:
            return 0
        return max(map(Tree._depth, node.children)) + 1

class BinaryTree(Tree):
    def __init__(self, tree):
        self.nodes = {}
        for id in range(tree.len):
            self.nodes[id] = Node(id)
        self.root = self.nodes[0]
        for id, node in tree.nodes.items():
            children = node.children
            pointer = self.nodes[id]
            for child in children:
                pointer = pointer.add_child(self.nodes[child.id])

tree = Tree()
for op in input():
    tree.move(op)
binary_tree = BinaryTree(tree)
print(f"{tree.depth()} => {binary_tree.depth()}")
```

代码运行截图

状态: Accepted

源代码

```
class Node:
    def __init__(self, id):
        self.id = id
        self.children = []
        self.father = None

    def add_child(self, child):
        self.children.append(child)
        child.father = self
        return child

class Tree:
    def __init__(self):
        self.root = Node(0)
        self.nodes = {0: self.root}
        self.len = 1
        self.pointer = self.root

    def move(self, op):
        if op == 'd':
            self.pointer = self.pointer.add_child(Node(self.len))
            self.nodes[self.len] = self.pointer
            self.len += 1
        if op == 'u':
            self.pointer = self.pointer.father

    def depth(self):
        return self._depth(self.root)

    @staticmethod
    def _depth(node):
        if len(node.children) == 0:
            return 0
        return max(map(Tree._depth, node.children)) + 1

class BinaryTree(Tree):
    def __init__(self, tree):
        self.nodes = {}
        for id in range(tree.len):
            self.nodes[id] = Node(id)
        self.root = self.nodes[0]
        for id, node in tree.nodes.items():
            children = node.children
            pointer = self.nodes[id]
            for child in children:
                pointer = pointer.add_child(self.nodes[child.id])

tree = Tree()
for op in input():
    tree.move(op)
binary_tree = BinaryTree(tree)
print(f"{tree.depth()} => {binary_tree.depth()}")
```

02255: 重建二叉树

http://cs101.openjudge.cn/practice/02255/

思路：复用`根据二叉树前中序序列建树`的代码，针对输入略作改动。

代码

```
class Node:
    def __init__(self, id, is_root=None):
        self.id = id
        self.left = None
        self.right = None
        self.father = None
        self.is_root = is_root

    def is_full(self):
        if self.id == '.':
            return True
        if self.left is None or self.right is None:
            return False
        return self.left.is_full() and self.right.is_full()

    def jump(self):
        if self.is_full() and self.is_root is None:
            return self.father.jump()
        return self

class BinaryTree:
    def __init__(self):
        self.nodes = {}
        self.root = None
        self.pointer = None
        self.placeholder = Node('.')

    def add_node(self, id):
        if id != '.':
            node = Node(id)
            if len(self.nodes) == 0:
                self.root = node
            else:
                self.pointer = self.pointer.jump()
                if self.pointer.left is None:
                    self.pointer.left = node
                    node.father = self.pointer
                elif self.pointer.right is None:
                    self.pointer.right = node
                    node.father = self.pointer
            self.pointer = node
```

```
            self.nodes[id] = node
        else:
            self.pointer = self.pointer.jump()
            if self.pointer.left is None:
                self.pointer.left = self.placeholder
            elif self.pointer.right is None:
                self.pointer.right = self.placeholder

    def middle_traverse(self):
        return self._middle_traverse(self.root)

    def backward_traverse(self):
        return self._backward_traverse(self.root)

    def _middle_traverse(self, node):
        if node == self.placeholder:
            return ''
        return self._middle_traverse(node.left) + node.id +
self._middle_traverse(node.right)

    def _backward_traverse(self, node):
        if node == self.placeholder:
            return ''
        return self._backward_traverse(node.left) +
self._backward_traverse(node.right) + node.id

binary_tree = BinaryTree()
for i in input():
    binary_tree.add_node(i)
print(binary_tree.middle_traverse())
print(binary_tree.backward_traverse())
```

代码运行截图

状态: Accepted

源代码

```
class Node:
    def __init__(self, id, is_root=None):
        self.id = id
        self.left = None
        self.right = None
        self.father = None
        self.is_root = is_root

    def is_full(self):
        if self.id == '.':
            return True
        if self.left is None or self.right is None:
            return False
        return self.left.is_full() and self.right.is_full()

    def jump(self):
        if self.is_full() and self.is_root is None:
            return self.father.jump()
        return self

class BinaryTree:
    def __init__(self):
        self.nodes = {}
        self.root = None
        self.pointer = None
        self.placeholder = Node('.')

    def add_node(self, id):
        if id != '.':
            node = Node(id)
            if len(self.nodes) == 0:
                self.root = node
            else:
                self.pointer = self.pointer.jump()
                if self.pointer.left is None:
                    self.pointer.left = node
                    node.father = self.pointer
                elif self.pointer.right is None:
                    self.pointer.right = node
                    node.father = self.pointer
            self.pointer = node
            self.nodes[id] = node
        else:
            self.pointer = self.pointer.jump()
            if self.pointer.left is None:
                self.pointer.left = self.placeholder
            elif self.pointer.right is None:
                self.pointer.right = self.placeholder

    def middle_traverse(self):
        return self._middle_traverse(self.root)

    def backward_traverse(self):
        return self._backward_traverse(self.root)
```

```
    def _middle_traverse(self, node):
        if node == self.placeholder:
            return ''
        return self._middle_traverse(node.left) + node.id + self._middle

    def _backward_traverse(self, node):
        if node == self.placeholder:
            return ''
        return self._backward_traverse(node.left) + self._backward_trave

binary_tree = BinaryTree()
for i in input():
    binary_tree.add_node(i)
print(binary_tree.middle_traverse())
print(binary_tree.backward_traverse())
```


01426: Find The Multiple

http://cs101.openjudge.cn/practice/01426/

要求用bfs实现

思路：后一层检查列表为遍历前一列表分别*10后+1或+0，以此构造bfs。

代码

```
1
def bfs(x:int, l:list=[1]):
2
    for y in l:
3
        if y % x == 0:
4
            return y
5
    l2 = []
6
    for y in l:
7
        l2.append(10*y + 0)
8
        l2.append(10*y + 1)
9
    return bfs(x, l2)
10
•
11
```

```
while True:
12
    x = int(input())
13
    if x != 0:
14
        print(bfs(x))
15
    else:
16
        break
```

状态: Accepted

源代码

```
class PigStack:
    stack:list

    def __init__(self):
        self.stack = []
    def push(self, weight):
        if not self.stack:
            self.stack.append((weight, weight))
        else:
            current_min = min(weight, self.stack[-1][1])
            self.stack.append((weight, current_min))
    def pop(self):
        if self.stack:
            self.stack.pop()
    def get_min(self):
        if self.stack:
            return self.stack[-1][1]

pig_stack = PigStack()
while True:
    try:
        inp = input().split()
        if inp[0] == "push":
            weight = int(inp[1])
            pig_stack.push(weight)
        elif inp[0] == "pop":
            pig_stack.pop()
        elif inp[0] == "min":
            min_weight = pig_stack.get_min()
            if min_weight is not None:
                print(min_weight)
    except EOFError:
        break
```

04115: 鸣人和佐助

bfs, http://cs101.openjudge.cn/practice/04115/

思路：比较暴力，存一下剩余的查克拉数目，避开曾在带有更多查克拉时已可经过的点。代码比较意识流，加了递归深度还是过了。

代码

```
1
import sys
2
sys.setrecursionlimit(100000)
3
•
4
m, n, t = map(int, input().split())
5
l = [[-1] * n for _ in range(m)]
6
move_offset = [(0, -1), (0, 1), (-1, 0), (1, 0)]
7
pos1 = ()
8
pos2 = ()
9
ll = []
10
for y in range(m):
11
    temp = input()
12
    if '@' in temp:
13
        x = temp.index('@')
14
        pos1 = (x, y)
15
        l[y][x] = t
16
    if '+' in temp:
17
        x = temp.index('+')
18
        pos2 = (x, y)
19
    ll.append(temp)
20
•
21
def bfs(lx:list, steps:int = 0):
22
```

```
    global pos2, move_offset, ll, l, n, m
23
    if lx == []:
24
        return -1
25
    l2 = []
26
    for info in lx:
27
        pos, t = info
28
        if pos == pos2:
29
            return steps
30
        else:
31
            x, y = pos
32
            for offset in move_offset:
33
                i, j = offset
34
                xi, yj, tt = x + i, y + j, t
35
                if xi >= 0 and yj >= 0 and xi < n and yj < m:
36
                    if ll[yj][xi] == '#':
37
                        tt -= 1
38
                    if tt >= 0 and l[yj][xi] < tt:
39
                        l[yj][xi] = tt
40
                        l2.append(((xi, yj), tt))
41
    return bfs(l2, steps + 1)
42
•
43
print(bfs([(pos1, t)]))
```

状态: Accepted

源代码

```
totalsteps = 0
momentum = 0
ans = 0
def zouni(passby:list, x:int, y:int, step:int):
    global totalsteps
    global momentum
    global ans
    if step == totalsteps:
        ans += 1
        return 0
    for i in momentum:
        if x+i[0] > -1 and x+i[0] < n and y+i[1] > -1 and y+i[1] < m:
            if passby[x+i[0]][y+i[1]] != 1:
                passby[x+i[0]][y+i[1]] = 1
                zouni(passby, x+i[0], y+i[1], step+1)
                passby[x+i[0]][y+i[1]] = 0
    else:
        return 0
momentum = [[1, 2], [-1, 2], [1, -2], [-1, -2], [2, 1], [-2, 1], [2, -1
for t in range(int(input())):
    n, m, x, y = map(int, input().split())
    totalsteps = n*m
    ans = 0
    passby = [[0]*m for _ in range(n)]
    passby[x][y] = 1
    zouni(passby, x, y, 1)
    print(ans)
```

20106: 走山路

Dijkstra, http://cs101.openjudge.cn/practice/20106/

思路：看了好久Dijkstra算法，才搞懂要怎么在题目里实现。维护一个与已经加入图的节点相邻的节点的前线heap，然后加入最近的节点并将其相邻节点heappush入表。这题和下一题的难点感觉都在于如何减枝（先加上权重还是先不加权重的问题），有了思路后一直在思考这个问题，看了一下群里发现有人写了一样思路的代码，就借鉴了一下大神的代码。

代码

```
1
from heapq import *
2
•
3
move_offset = [(0, -1), (0, 1), (-1, 0), (1, 0)]
```

```
4
inf = float('inf')
5
m, n, p = map(int, input().split())
6
l = [list(map(lambda x: inf if x == '#' else int(x), input().split())) for _
in range(m)]
7
•
8
•
9
def bfs(x0, y0, xt, yt):
10
    global l, m, n, move_offset, inf
11
    distance = [[inf] * n for _ in range(m)]
12
    if l[y0][x0] == inf or l[yt][xt] == inf: return 'NO'
13
    distance[y0][x0] = 0
14
    front = [(0, x0, y0)]
15
    while front:
16
        d, x, y = heappop(front)
17
        if (x, y) == (xt, yt):
18
            return d
19
        h = l[y][x]
20
        for movement in move_offset:
21
            i, j = movement
22
            xn, yn = x + i, y + j
23
            if 0 <= xn < n and 0 <= yn < m and l[yn][xn] != inf:
24
                dn = abs(l[yn][xn] - h) + d
25
                if distance[yn][xn] > dn:
26
                    distance[yn][xn] = dn
27
                    heappush(front, (dn, xn, yn))
28
    return 'NO'
29
•
30
for _ in range(p):
31
```

```
    y0, x0, yt, xt = map(int, input().split())
32
    print(bfs(x0, y0, xt, yt))
```

状态: Accepted

源代码

```
from queue import deque

class Vertex:
    id:int
    word:str
    connected:list
    visited:bool

    def __init__(self, id, word):
        self.id = id
        self.word = word
        self.connected = []
        self.visited = False
        self.father = None

class Graph:
    words:list
    word2vertex:dict
    len:int

    def __init__(self, words):
        self.words = words
        self.word2vertex = {}
        self.len = len(words)
        for id, word in enumerate(words):
            self.word2vertex[word] = Vertex(id, word)
    def add_edge(self, word1, word2):
        self.word2vertex[word1].connected.append(self.word2vertex[word2]
        self.word2vertex[word2].connected.append(self.word2vertex[word1]
    def connect(self):
        buckets = {}
        for word in self.words:
            for i, _ in enumerate(word):
                bucket = f"{word[:i]}_{word[i + 1:]}"
                buckets.setdefault(bucket, set()).add(word)
        for similar_words in buckets.values():
            for word1 in similar_words:
                for word2 in similar_words - {word1}:
                    self.add_edge(word1, word2)
    def search(self, start, end):
        queue = deque()
        queue.append(self.word2vertex[start])
        self.word2vertex[start].visited = True
        while len(queue) != 0:
            now = queue.popleft()
            if now == self.word2vertex[end]:
                break
            for child in now.connected:
                if child.visited == False:
                    queue.append(child)
                    child.father = now
                    child.visited = True
        if self.word2vertex[end].father is None:
            return "NO"
        res = [end]
```

```
        now = self.word2vertex[end]
        while now.father is not None:
            res.append(now.father.word)
            now = now.father
        return ' '.join(reversed(res))

n = int(input())
graph = Graph([input() for _ in range(n)])
graph.connect()
print(graph.search(*input().split()))
```


05442: 兔子与星空

Prim, http://cs101.openjudge.cn/practice/05442/

思路：暴力枚举，每次加入距离图最近的一条边。

代码

```
1
n = int(input())
2
dic = dict()
3
nodes = list()
4
ans = 0
5
for _ in range(n-1):
6
    raw = list(input().split())
7
    i = 0
8
    x = ''
9
    key = ''
10
    for char in raw:
11
        if i > 1:
12
            if i % 2 == 0:
13
                if char not in nodes:
```

```
14                      nodes.append(char)
15                  key = x + char
16              else:
17                  dic[key] = int(char)
18          elif i == 1:
19              pass
20          elif i == 0:
21              if char not in nodes:
22                  nodes.append(char)
23              x = char
24          i += 1
25  cnodes = []
26  cnodes.append(nodes.pop())
27  while True:
28      mweight = 100
29      medge = ''
30      onode = ''
31      monode = ''
32      cweight = ''
33      for edge in dic.keys():
34          for cnode in cnodes:
35              if cnode in edge:
36                  onode = edge.strip(cnode)
37                  if onode not in cnodes:
38                      cweight = dic[edge]
39                      if cweight < mweight:
40                          medge = edge
41                          mweight = cweight
```

```
42
                        monode = onode
43
    nodes.remove(monode)
44
    cnodes.append(monode)
45
    ans += mweight
46
    if not nodes:
47
        break
48
print(ans)
```

状态：Accepted

源代码

```
directions = [(1, 2), (2, 1), (1, -2), (2, -1), (-1, 2), (-2, 1), (-1, -
def dfs(n, x, y, traversed):
    if n % 2 == 1:
        return True
    if len(traversed) == n ** 2:
        return True
    for dx, dy in directions:
        xx, yy = x + dx, y + dy
        if (xx, yy) in traversed or xx < 0 or xx >= n or yy < 0 or yy >=
            continue
        if dfs(n, xx, yy, traversed + [(xx, yy)]):
            return True
    return False

n = int(input())
x, y = map(int, input().split())
print("success" if dfs(n, x, y, [(x, y)]) else "fail")
```

2. 学习总结和收获

感觉这周题目还是有一定难度的，尤其是bfs的剪枝问题。这块在期末复习时候需要关注一下。