

Assignment #A: 图论：遍历，树算及栈

Updated 2018 GMT+8 Apr 21, 2024

2024 spring, Compiled by 天幕 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

```
1 class Node:
2     def __init__(self, id):
3         self.id = id
4         self.children = []
5         self.father = None
6
7     def add_child(self, child):
8         self.children.append(child)
9         child.father = self
10        return child
11
12 class Tree:
13     def __init__(self):
14         self.root = Node(0)
```

```

15         self.nodes = {0: self.root}
16         self.len = 1
17         self.pointer = self.root
18
19     def move(self, op):
20         if op == 'd':
21             self.pointer = self.pointer.add_child(Node(self.len))
22             self.nodes[self.len] = self.pointer
23             self.len += 1
24         if op == 'u':
25             self.pointer = self.pointer.father
26
27     def depth(self):
28         return self._depth(self.root)
29
30     @staticmethod
31     def _depth(node):
32         if len(node.children) == 0:
33             return 0
34         return max(map(Tree._depth, node.children)) + 1
35
36 class BinaryTree(Tree):
37     def __init__(self, tree):
38         self.nodes = {}
39         for id in range(tree.len):
40             self.nodes[id] = Node(id)
41         self.root = self.nodes[0]
42         for id, node in tree.nodes.items():
43             children = node.children
44             pointer = self.nodes[id]
45             for child in children:
46                 pointer = pointer.add_child(self.nodes[child.id])
47
48 tree = Tree()
49 for op in input():
50     tree.move(op)
51 binary_tree = BinaryTree(tree)
52 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

```
1 class Node:
2     def __init__(self, id, is_root=None):
3         self.id = id
4         self.left = None
5         self.right = None
6         self.father = None
7         self.is_root = is_root
8
9     def is_full(self):
10        if self.id == '.':
11            return True
12        if self.left is None or self.right is None:
13            return False
14        return self.left.is_full() and self.right.is_full()
15
16    def jump(self):
17        if self.is_full() and self.is_root is None:
18            return self.father.jump()
19        return self
20
21 class BinaryTree:
22     def __init__(self):
23         self.nodes = {}
24         self.root = None
25         self.pointer = None
26         self.placeholder = Node('.')
27
28     def add_node(self, id):
29         if id != '.':
30             node = Node(id)
31             if len(self.nodes) == 0:
32                 self.root = node
33             else:
34                 self.pointer = self.pointer.jump()
35                 if self.pointer.left is None:
36                     self.pointer.left = node
37                     node.father = self.pointer
38                 elif self.pointer.right is None:
39                     self.pointer.right = node
40                     node.father = self.pointer
41                 self.pointer = node
```

```
42         self.nodes[id] = node
43     else:
44         self.pointer = self.pointer.jump()
45         if self.pointer.left is None:
46             self.pointer.left = self.placeholder
47         elif self.pointer.right is None:
48             self.pointer.right = self.placeholder
49
50     def middle_traverse(self):
51         return self._middle_traverse(self.root)
52
53     def backward_traverse(self):
54         return self._backward_traverse(self.root)
55
56     def _middle_traverse(self, node):
57         if node == self.placeholder:
58             return ''
59         return self._middle_traverse(node.left) + node.id +
self._middle_traverse(node.right)
60
61     def _backward_traverse(self, node):
62         if node == self.placeholder:
63             return ''
64         return self._backward_traverse(node.left) +
self._backward_traverse(node.right) + node.id
65
66 binary_tree = BinaryTree()
67 for i in input():
68     binary_tree.add_node(i)
69 print(binary_tree.middle_traverse())
70 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_traverse

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

```

©2002-2022 POJ 京ICP备20010980号-1

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 1
2 def bfs(x:int, l:list=[1]):
3     2
4     for y in l:
5         3
6         if y % x == 0:
7             4
8             return y
9         5
10    l2 = []
11    6
12    for y in l:
13        7
14        l2.append(10*y + 0)
15        8
16        l2.append(10*y + 1)
17    9
18    return bfs(x, l2)
19 10
20 •
21 11

```

```

22 while True:
23     12
24     x = int(input())
25     13
26     if x != 0:
27         14
28         print(bfs(x))
29     15
30     else:
31         16
32         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 1
2 import sys
3 2
4 sys.setrecursionlimit(100000)
5 3
6 •
7 4
8 m, n, t = map(int, input().split())
9 5
10 l = [[-1] * n for _ in range(m)]
11 6
12 move_offset = [(0, -1), (0, 1), (-1, 0), (1, 0)]
13 7
14 pos1 = ()
15 8
16 pos2 = ()
17 9
18 ll = []
19 10
20 for y in range(m):
21 11
22     temp = input()
23 12
24     if '@' in temp:
25 13
26         x = temp.index('@')
27 14
28         pos1 = (x, y)
29 15
30         l[y][x] = t
31 16
32     if '+' in temp:
33 17
34         x = temp.index('+')
35 18
36         pos2 = (x, y)
37 19
38     ll.append(temp)
39 20
40 •
41 21
42 def bfs(lx:list, steps:int = 0):
43 22
```

```

44     global pos2, move_offset, l1, l, n, m
45 23
46     if lx == []:
47 24
48         return -1
49 25
50     l2 = []
51 26
52     for info in lx:
53 27
54         pos, t = info
55 28
56         if pos == pos2:
57 29
58             return steps
59 30
60         else:
61 31
62             x, y = pos
63 32
64             for offset in move_offset:
65 33
66                 i, j = offset
67 34
68                 xi, yj, tt = x + i, y + j, t
69 35
70                 if xi >= 0 and yj >= 0 and xi < n and yj < m:
71 36
72                     if l1[yj][xi] == '#':
73 37
74                         tt -= 1
75 38
76                     if tt >= 0 and l[yj][xi] < tt:
77 39
78                         l[yj][xi] = tt
79 40
80                         l2.append(((xi, yj), tt))
81 41
82     return bfs(l2, steps + 1)
83 42
84 •
85 43
86 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 1
2 from heapq import *
3 2
4 •
5 3
6 move_offset = [(0, -1), (0, 1), (-1, 0), (1, 0)]
```

```

7 4
8 inf = float('inf')
9 5
10 m, n, p = map(int, input().split())
11 6
12 l = [list(map(lambda x: inf if x == '#' else int(x), input().split())) for _
    in range(m)]
13 7
14 •
15 8
16 •
17 9
18 def bfs(x0, y0, xt, yt):
19 10
20     global l, m, n, move_offset, inf
21 11
22     distance = [[inf] * n for _ in range(m)]
23 12
24     if l[y0][x0] == inf or l[yt][xt] == inf: return 'NO'
25 13
26     distance[y0][x0] = 0
27 14
28     front = [(0, x0, y0)]
29 15
30     while front:
31 16
32         d, x, y = heappop(front)
33 17
34         if (x, y) == (xt, yt):
35 18
36             return d
37 19
38         h = l[y][x]
39 20
40         for movement in move_offset:
41 21
42             i, j = movement
43 22
44             xn, yn = x + i, y + j
45 23
46             if 0 <= xn < n and 0 <= yn < m and l[yn][xn] != inf:
47 24
48                 dn = abs(l[yn][xn] - h) + d
49 25
50                 if distance[yn][xn] > dn:
51 26
52                     distance[yn][xn] = dn
53 27
54                     heappush(front, (dn, xn, yn))
55 28
56     return 'NO'
57 29
58 •
59 30
60 for _ in range(p):
61 31

```

```
62     y0, x0, yt, xt = map(int, input().split())
63     32
64     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()))

```

©2002-2022 POJ 京ICP备20010980号-1

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  1
2  n = int(input())
3  2
4  dic = dict()
5  3
6  nodes = list()
7  4
8  ans = 0
9  5
10 for _ in range(n-1):
11  6
12     raw = list(input().split())
13  7
14     i = 0
15  8
16     x = ''
17  9
18     key = ''
19  10
20     for char in raw:
21  11
22         if i > 1:
23  12
24             if i % 2 == 0:
25  13
26                 if char not in nodes:

```

```

27 14
28         nodes.append(char)
29 15
30         key = x + char
31 16
32         else:
33 17
34             dic[key] = int(char)
35 18
36         elif i == 1:
37 19
38             pass
39 20
40         elif i == 0:
41 21
42             if char not in nodes:
43 22
44                 nodes.append(char)
45 23
46                 x = char
47 24
48                 i += 1
49 25
50 cnodes = []
51 26
52 cnodes.append(nodes.pop())
53 27
54 while True:
55 28
56     mweight = 100
57 29
58     medge = ''
59 30
60     onode = ''
61 31
62     monode = ''
63 32
64     cweight = ''
65 33
66     for edge in dic.keys():
67 34
68         for cnode in cnodes:
69 35
70             if cnode in edge:
71 36
72                 onode = edge.strip(cnode)
73 37
74                 if onode not in cnodes:
75 38
76                     cweight = dic[edge]
77 39
78                     if cweight < mweight:
79 40
80                         medge = edge
81 41
82                         mweight = cweight

```



```

83 42
84         monode = onode
85 43
86     nodes.remove(monode)
87 44
88     cnodes.append(monode)
89 45
90     ans += mweight
91 46
92     if not nodes:
93 47
94         break
95 48
96 print(ans)

```

状态: Accepted

源代码

```

directions = [(1, 2), (2, 1), (1, -2), (2, -1), (-1, 2), (-2, 1), (-1, -2), (-2, -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 >= n:
            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。