

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 def decompile(string: str):
2     string = '((' + string + '))'
3     stack = []
4     for char in string:
5         if char == ')':
6             temp = []
7             while 1:
8                 try:
9                     x = stack.pop()
10                    if x == '(':
11                        raise EOFError
12                    else:
13                        temp.append(x)
14            except EOFError:
```

```

15         break
16         stack += temp
17     else:
18         stack.append(char)
19     return ''.join(stack)
20 print(decompile(input()))

```

代码运行截图

状态: Accepted

源代码

```

def decompile(string: str):
    string = '(' + string + ')'
    stack = []
    for char in string:
        if char == ')':
            temp = []
            while 1:
                try:
                    x = stack.pop()
                    if x == '(':
                        raise EOFError
                    else:
                        temp.append(x)
                except EOFError:
                    break
            stack += temp
        else:
            stack.append(char)
    return ''.join(stack)
print(decompile(input()))

```

©2002-2022 POJ 京ICP备20010980号-1

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(object):
2     _ID = 0
3     NodeID: int
4     pNodeID: int
5     name: str

```

```

6     sub:list      #List<Node>
7     depth:int
8     def __init__(self, name, l, pNodeID= -1, depth:int=0):
9         self.NodeID = self._ID
10        self.__class__._ID += 1
11        self.pNodeID = pNodeID
12        self.sub = []
13        self.name = name
14        self.depth = depth
15        for node in l:
16            self.sub.append(node)
17    def info(self):
18        return (self.NodeID, self.sub)
19
20    class Tree(object):
21        tree:dict
22        root:Node
23        def __init__(self):
24            self.tree = dict()
25            self.root = None
26
27        def add(self, node:Node):
28            cNodeID, cSubNodes = node.info()
29
30            self.tree[cNodeID] = node    #加入树
31
32            if not self.root:    #尝试转移根节点
33                self.root = node
34            elif self.get(self.root.NodeID) in cSubNodes:
35                self.root = node
36
37            for nodes in self.tree.values():    #尝试添加父节点
38                aNodeID, aSubNodes = nodes.info()
39                if self.get(cNodeID) in aSubNodes:    #是子节点
40                    node.pNodeID = aNodeID
41        def get(self, nodeID):
42            if nodeID == -1:
43                return False
44            else:
45                return self.tree[nodeID]
46        def getDepth(self, node:Node):
47            cSubNodes = node.sub
48            if cSubNodes:
49                if node.depth == 0:
50                    self.depth = 1 + max([self.getDepth(subNode) for subNode in
cSubNodes])
51                return node.depth
52            return 0
53        def getTreeDep(self):    # This can also init the tree
54            return self.getDepth(self.root)
55
56        def levelOrderFrom(self, node:Node):
57            if not node: return []
58
59            res, queue = [], [node]
60            while queue:

```

```

61         level_node = []
62
63         for _ in range(len(queue)):
64             node = queue.pop(0)
65             level_node.append(node.name)
66
67             for x in node.sub:
68                 if x:
69                     queue.append(x)
70         res.append(level_node)
71
72         return "".join(["".join(x) for x in res])
73
74     def levelOrder(self):
75         return self.levelOrderFrom(self.root)
76     def preOrderFrom(self, node:Node): # 先序遍历
77         if not node: return ''
78         return node.name + "".join([self.preOrderFrom(x) for x in
node.sub])
79     def preOrder(self):
80         return self.preOrderFrom(self.root)
81     def postOrderFrom(self, node:Node): # 后序遍历
82         if not node: return ''
83         return "".join([self.postOrderFrom(x) for x in node.sub]) +
node.name
84     def postOrder(self):
85         return self.postOrderFrom(self.root)
86
87
88     def toTree(preOrPost: str, middle: str, index: int) -> Tree:
89         def toNode(tree: Tree, middle: str, preOrPost: str, index: int):
90             try:
91                 rootName = preOrPost[index]
92                 rootIndex = middle.find(rootName)
93                 info = middle[:rootIndex], middle[rootIndex + 1:],
preOrPost[1:rootIndex + 1], preOrPost[rootIndex + 1:]
94             except IndexError:
95                 return False
96             if info == ('', '', '', ''):
97                 node = Node(rootName, [])
98                 tree.add(node)
99                 return(node)
100             lSubTreeMiddle, rSubTreeMiddle, lSubTreePreOrPost,
rSubTreePreOrPost = info
101             node = Node(rootName, [toNode(tree, lSubTreeMiddle,
lSubTreePreOrPost, index), toNode(tree, rSubTreeMiddle, rSubTreePreOrPost,
index)])
102             tree.add(node)
103             return(node)
104         myTree = Tree()
105         toNode(myTree, middle, preOrPost, index)
106         return myTree
107
108     while True:
109         try:
110             print(toTree(*input().split(), 0).postOrder())

```

```
111     except EOFError:
112         break
```

代码运行截图

```

def levelOrderFrom(self, node:Node):
    if not node: return []

    res, queue = [], [node]
    while queue:
        level_node = []

        for _ in range(len(queue)):
            node = queue.pop(0)
            level_node.append(node.name)

            for x in node.sub:
                if x:
                    queue.append(x)
        res.append(level_node)

    return "".join(["".join(x) for x in res])

def levelOrder(self):
    return self.levelOrderFrom(self.root)
def preOrderFrom(self, node:Node): # 先序遍历
    if not node: return ''
    return node.name + "".join([self.preOrderFrom(x) for x in node.sub])
def preOrder(self):
    return self.preOrderFrom(self.root)
def postOrderFrom(self, node:Node): # 后序遍历
    if not node: return ''
    return "".join([self.postOrderFrom(x) for x in node.sub]) + node.name
def postOrder(self):
    return self.postOrderFrom(self.root)

def toTree(preOrPost: str, middle: str, index: int) -> Tree:
    def toNode(tree: Tree, middle: str, preOrPost: str, index: int):
        try:
            rootName = preOrPost[index]
            rootIndex = middle.find(rootName)
            info = middle[:rootIndex], middle[rootIndex + 1:], preOrPost[index + 1:]
        except IndexError:
            return False
        if info == ('', '', '', ''):
            node = Node(rootName, [])
            tree.add(node)
            return (node)
        lSubTreeMiddle, rSubTreeMiddle, lSubTreePreOrPost, rSubTreePreOrPost = info
        node = Node(rootName, [toNode(tree, lSubTreeMiddle, lSubTreePreOrPost, 0),
                               toNode(tree, rSubTreeMiddle, rSubTreePreOrPost, 0)])
        tree.add(node)
        return (node)
    myTree = Tree()
    toNode(myTree, middle, preOrPost, index)
    return myTree

while True:
    try:
        print(toTree(*input().split(), 0).postOrder())
    except EOFError:
        break

```

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
```

代码运行截图

状态: Accepted

源代码

```
def bfs(x:int, l:list=[1]):
    for y in l:
        if y % x == 0:
            return y
    l2 = []
    for y in l:
        l2.append(10*y + 0)
        l2.append(10*y + 1)
    return bfs(x, l2)

while True:
    x = int(input())
    if x != 0:
        print(bfs(x))
    else:
        break
```

©2002-2022 POJ 京ICP备20010980号-1

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     l1.append(temp)
20
21 def bfs(lx:list, steps:int = 0):
22     global pos2, move_offset, l1, 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 l1[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)]))
```

代码运行截图

状态: Accepted

源代码

```
import sys
sys.setrecursionlimit(100000)

m, n, t = map(int, input().split())
l = [[-1] * n for _ in range(m)]
move_offset = [(0, -1), (0, 1), (-1, 0), (1, 0)]
pos1 = ()
pos2 = ()
ll = []
for y in range(m):
    temp = input()
    if '@' in temp:
        x = temp.index('@')
        pos1 = (x, y)
        l[y][x] = t
    if '+' in temp:
        x = temp.index('+')
        pos2 = (x, y)
    ll.append(temp)

def bfs(lx:list, steps:int = 0):
    global pos2, move_offset, ll, l, n, m
    if lx == []:
        return -1
    l2 = []
    for info in lx:
        pos, t = info
        if pos == pos2:
            return steps
        else:
            x, y = pos
            for offset in move_offset:
                i, j = offset
                xi, yj, tt = x + i, y + j, t
                if xi >= 0 and yj >= 0 and xi < n and yj < m:
                    if ll[yj][xi] == '#':
                        tt -= 1
                    if tt >= 0 and l[yj][xi] < tt:
                        l[yj][xi] = tt
                        l2.append(((xi, yj), tt))
    return bfs(l2, steps + 1)

print(bfs([(pos1, t)]))
```

©2002-2022 POJ 京ICP备20010980号-1

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))
```

代码运行截图

状态: Accepted

源代码

```
from heapq import *

move_offset = [(0, -1), (0, 1), (-1, 0), (1, 0)]

inf = float('inf')
m, n, p = map(int, input().split())
l = [list(map(lambda x: inf if x == '#' else int(x), input().split())) for _ in range(m)]

def bfs(x0, y0, xt, yt):
    global l, m, n, move_offset, inf
    distance = [[inf] * n for _ in range(m)]
    if l[y0][x0] == inf or l[yt][xt] == inf: return 'NO'
    distance[y0][x0] = 0
    front = [(0, x0, y0)]
    while front:
        d, x, y = heappop(front)
        if (x, y) == (xt, yt):
            return d
        h = l[y][x]
        for movement in move_offset:
            i, j = movement
            xn, yn = x + i, y + j
            node = (xn, yn)

            if 0 <= xn < n and 0 <= yn < m and l[yn][xn] != inf:
                dn = abs(l[yn][xn] - h) + d
                if distance[yn][xn] > dn:
                    distance[yn][xn] = dn
                    heappush(front, (dn, xn, yn))

    return 'NO'

for _ in range(p):
    y0, x0, yt, xt = map(int, input().split())
    print(bfs(x0, y0, xt, yt))
```

©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 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)

```

代码运行截图

状态: Accepted

源代码

```
n = int(input())
dic = dict()
nodes = list()
ans = 0
for _ in range(n-1):
    raw = list(input().split())
    i = 0
    x = ''
    key = ''
    for char in raw:
        if i > 1:
            if i % 2 == 0:
                if char not in nodes:
                    nodes.append(char)
                    key = x + char
            else:
                dic[key] = int(char)
        elif i == 1:
            pass
        elif i == 0:
            if char not in nodes:
                nodes.append(char)
            x = char
        i += 1
    cnodes = []
    cnodes.append(nodes.pop())
    while True:
        mweight = 100
        medge = ''
        onode = ''
        monode = ''
        cweight = ''
        for edge in dic.keys():
            for cnode in cnodes:
                if cnode in edge:
                    onode = edge.strip(cnode)
                    if onode not in cnodes:
                        cweight = dic[edge]
                        if cweight < mweight:
                            medge = edge
                            mweight = cweight
                            monode = onode
        nodes.remove(monode)
        cnodes.append(monode)
        ans += mweight
        if not nodes:
            break
    print(ans)
```

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。