Assignment #5: "树"算：概念、表示、解析、遍历

Updated 2124 GMT+8 March 17, 2024

2024 spring, Complied by ==同学的姓名、院系==

说明：

1）The complete process to learn DSA from scratch can be broken into 4 parts:

Learn about Time complexities, learn the basics of individual Data Structures, learn the basics of Algorithms, and practice Problems.

2）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。

3）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。

4）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

27638: 求二叉树的高度和叶子数目

http://cs101.openjudge.cn/practice/27638/

思路：

建立树时直接找出叶子，递归求每个叶子高度最大值输出。

代码

```
```

```
from collections import defaultdict

class Node:
    tree = defaultdict()
    def __init__(self, key):
        self.key = key
        self.father = None
        self.children = []
        self.height = -1
        Node.tree[self.key] = self

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

    def __str__(self):
        return self.key

    def check_height(self):
        if self.father is None:
            return 0
        if self.height != -1:
            return self.height
        return self.father.check_height() + 1

root = 0
leaves = []

n = int(input())
for index in range(n):
    node = Node(index) if index not in Node.tree else Node.tree[index]
    l, r = map(int, input().split())
    if l == r == -1:
        leaves.append(node)
    else:
        for i in (l, r):
            if i != -1:
                node.add_child(Node(i) if i not in Node.tree else Node.tree[i])

print(max(x.check_height() for x in leaves), len(leaves))
```

代码运行截图 ==（至少包含有"Accepted"）==

状态: Accepted

源代码

```
#27638:求二叉树的高度和叶子数目
from collections import defaultdict

class Node:
    tree = defaultdict()
    def __init__(self, key):
        self.key = key
        self.father = None
        self.children = []
        self.height = -1
        Node.tree[self.key] = self

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

    def __str__(self):
        return self.key

    def check_height(self):
        if self.father is None:
            return 0
        if self.height != -1:
            return self.height
        return self.father.check_height() + 1

root = 0
leaves = []

n = int(input())
for index in range(n):
    node = Node(index) if index not in Node.tree else Node.tree[index]
    l, r = map(int, input().split())
    if l == r == -1:
        leaves.append(node)
    else:
        for i in (l, r):
            if i != -1:
                node.add_child(Node(i) if i not in Node.tree else Node.t

print(max(x.check_height() for x in leaves), len(leaves))
```

24729: 括号嵌套树

http://cs101.openjudge.cn/practice/24729/

思路：

对于繁琐的括号parser写了一个模板，实现树结构之后从输入递归建树，对于前序和后序输出采用相同的思路递归处理。

代码

```
from collections import defaultdict

class Node:
    tree = defaultdict()
    def __init__(self, key, children):
        self.key = key
        Node.tree[self.key] = self
        if children is None:
            self.children = []
        else:
            self.children = children

    def __str__(self):
        return self.key

def parse_parentheses(s):
    def push(obj, l, depth):
        while depth:
            l = l[-1]
            depth -= 1
        l.append(obj)

    groups = []
    depth = 0
    try:
        for char in s:
            if char == '(':
                push([], groups, depth)
                depth += 1
            elif char == ')':
                depth -= 1
            elif char != ',':
                push(char, groups, depth)
    except IndexError as exc:
        raise ValueError('Parentheses mismatch') from exc
    if depth > 0:
        raise ValueError('Parentheses mismatch')
    else:
        return groups
```

```
def parse_nodes(l):
    res = []
    for index, node in enumerate(l):
        if isinstance(node, str):
            if index + 1 <= len(l) - 1 and isinstance(l[index + 1], list):
                res.append(Node(node, parse_nodes(l[index + 1])))
            else:
                res.append(Node(node, []))
    return res

def forward(*node):
    res = []
    for i in node:
        res.append(str(i))
        for j in forward(*i.children):
            res.append(j)
    return res

def backward(*node):
    res = []
    for i in node:
        for j in backward(*i.children):
            res.append(j)
        res.append(str(i))
    return res

parsed = parse_parentheses(input())
parse_nodes(parsed)
print(''.join(forward(Node.tree[parsed[0]])))
print(''.join(backward(Node.tree[parsed[0]])))
```

代码运行截图 ==（至少包含有"Accepted"）==

02775: 文件结构“图”

http://cs101.openjudge.cn/practice/02775/

思路：

递归返回不同层级目录的字符串，使用栈标记当前的目录层级。

代码

```
from collections import defaultdict

class Dir:
```

```
    tree = defaultdict()
    def __init__(self, dir_name, dss):
        self.ds = dss
        self.name = dir_name
        Dir.tree[self.name] = self
        self.files = []
        self.dirs = []
        self.depth = -1

    def __str__(self):
        prefix = "|     " * self.depth
        res = prefix + self.name + "\n"
        for dirr in self.dirs:
            res += str(dirr)
        for file in sorted(self.files):
            res += (prefix + file + "\n")
        return res

    def add_dir(self, dirr):
        self.dirs.append(dirr)
        dirr.depth = self.depth + 1

    def add_file(self, file):
        self.files.append(file)

ds = 1
a = 2
while 1:
    Dir("ROOT", ds)
    stack = [Dir.tree["ROOT"]]
    stack[-1].depth = 0
    while 1:
        current = input()
        if current in ('#', '*'):
            if current == '#':
                a = 0
            else:
                a = 2
            break
        elif a == 2 and current != '*':
            a = 1
            print(f"DATA SET {ds}:")
        if current[0] == 'd':
            current_dir = Dir(current, ds)
            stack[-1].add_dir(current_dir)
            stack.append(current_dir)
        if current == ']':
            stack.pop()
        if current[0] == 'f':
            stack[-1].add_file(current)
    if a == 0:
        break
```

```
    print(Dir.tree["ROOT"])
    ds += 1
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

25140: 根据后序表达式建立队列表达式

http://cs101.openjudge.cn/practice/25140/

思路：

使用栈处理输入，递归遍历树的每一层。

代码

```
class Repr:
    def __init__(self, op, arg1, arg2):
        self.op, self.arg1, self.arg2 = op, arg1, arg2

def parse_input(l):
    stack = []
    for item in l:
        if item.islower():
            stack.append(item)
        else:
            stack.append(Repr(item, stack.pop(), stack.pop()))
    return stack[-1]

def parse_tree(repr_list):
    if repr_list == []:
        return ''
    res = ''
    stack = []
    for repr_ in repr_list:
        if isinstance(repr_, Repr):
            res += repr_.op
            stack.append(repr_.arg2)
            stack.append(repr_.arg1)
        else:
            res += repr_
    return res + parse_tree(stack)

n = int(input())
for _ in range(n):
    my_repr = parse_input(input())
    print(parse_tree([my_repr])[::-1])
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

24750: 根据二叉树中后序序列建树

http://cs101.openjudge.cn/practice/24750/

思路：

直接按顺序输出递归结果，不用实现树结构。

代码

```
def parse(mid, post):
    if mid == '':
        return ''
    if len(mid) == 1:
        return mid[0]
    root = post[-1]
    root_index = mid.find(root)
    return root + parse(mid[:root_index], post[:root_index]) + parse(mid[root_index + 1:],
post[root_index:-1])

print(parse(input(), input()))
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

状态: Accepted

源代码

```
#24750:根据二叉树中后序序列建树
def parse(mid, post):
    if mid == '':
        return ''
    if len(mid) == 1:
        return mid[0]
    root = post[-1]
    root_index = mid.find(root)
    return root + parse(mid[:root_index], post[:root_index]) + parse(mi

print(parse(input(), input()))
```


22158: 根据二叉树前中序序列建树

http://cs101.openjudge.cn/practice/22158/

思路:

同样思路。

代码

```
def parse(pre, mid):
    if mid == '':
        return ''
    if len(mid) == 1:
        return mid[0]
    root = pre[0]
    root_index = mid.find(root)
    return parse(pre[1:root_index + 1], mid[:root_index]) + parse(pre[root_index + 1:],
mid[root_index + 1:]) + root

while 1:
    try:
        print(parse(input(), input()))
    except EOFError:
        break
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

2. 学习总结和收获

在不考虑性能开销的情况下，善用递归可以获得优雅的实现，这也非常符合函数式编程的思路。太伟大了递归!