Assignment #6: "树"算：Huffman,BinHeap,BST,AVL,DisjointSet

Updated 2214 GMT+8 March 24, 2024

2024 spring, Complied by 刘子暄 环境科学与工程学院

说明：

1）这次作业内容不简单，耗时长的话直接参考题解。

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：Windows 11

Python编程环境: PyCharm Community Edition 2023.3

1. 题目

22275: 二叉搜索树的遍历

http://cs101.openjudge.cn/practice/22275/

思路：与二叉树相同点：
节点定义，后序遍历输出函数（将左子树和右子树看作节点，执行超级操作）
注意点：

建树和遍历时都要考虑到空树情况，这是大部分树的基本情况

代码

```python
class TreeNode:
    def __init__(self,val):
        self.val = val
        self.left = None
        self.right = None

def build_tree(pre):
    max = len(pre)
    if max == 0:
        return None#最终边界条件

    root = TreeNode(pre[0])

    id = max#可能出现全为左树的情况
    for i in range(1, max):
        if pre[i] > root.val:
            id = i
            break
    root.left = build_tree(pre[1:id])
    root.right = build_tree(pre[id:])

    return root

def postorder(root):
    if root is None:
        return []
    out = []
    out.extend(postorder(root.left))
    out.extend(postorder(root.right))
    out.append(str(root.val))#join只能链接字符串格式

    return out#不能直接在这里join，会出现递归时空格，空格数增加

n = int(input())
preorder = list(map(int,input().split()))
print(' '.join(postorder(build_tree(preorder))))
```

代码运行截图 （至少包含有"Accepted"）

#44497281提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
class TreeNode:
    def __init__(self,val):
        self.val = val
        self.left = None
        self.right = None

def build_tree(pre):
    max = len(pre)
    if max == 0:
        return None#最终边界条件

    root = TreeNode(pre[0])

    id = max#可能出现全为左树的情况
    for i in range(1, max):
        if pre[i] > root.val:
            id = i
            break
    root.left = build_tree(pre[1:id])
    root.right = build_tree(pre[id:])

    return root

def postorder(root):
    if root is None:
        return []
    out = []
    out.extend(postorder(root.left))
    out.extend(postorder(root.right))
    out.append(str(root.val))#join只能链接字符串格式

    return out#不能直接在这里join，会出现递归时空格，空格数增加

n = int(input())
preorder = list(map(int,input().split()))
print(' '.join(postorder(build_tree(preorder))))
```

基本信息

#:	44497281
题目:	22275
提交人:	刘子暄
内存:	4036kB
时间:	25ms
语言:	Python3
提交时间:	2024-04-01 16:51:58

05455: 二叉搜索树的层次遍历

http://cs101.openjudge.cn/practice/05455/

思路：建树过程是将每一个node从root开始和每一个node比较，最后找到合适的位置insert

层次遍历时，用两个列表来存数据，分别叫做节点组和结果组，当节点组不为空，从前弹出节点，结果组加入节点val，将节点左子和右子加入节点组

代码

```python
class TreeNode:
    def __init__(self,val):
        self.val = val
        self.left = None
        self.right = None

def build_tree_insert(node,other_node_val):
    if node is None:
        return TreeNode(other_node_val)#None是节点的边界条件
    if other_node_val < node.val:
        node.left = build_tree_insert(node.left, other_node_val)
    elif other_node_val > node.val:
        node.right = build_tree_insert(node.right, other_node_val)
    return node#直观解释就是将现在的root与每一个node比较，判断放入做自主还是右子树，之后递归此操作



def level_order_traversal(root):
    queue = [root]
    traversal = []
    while queue:
        node = queue.pop(0)
        traversal.append(str(node.val))
        if node.left:
            queue.append(node.left)
        if node.right:
            queue.append(node.right)
    return traversal

n = list(map(int, input().strip().split()))
nodes = list(dict.fromkeys(n))#使用set不稳定
root = None
for i in nodes:
    root = build_tree_insert(root, i)
print(' '.join(level_order_traversal(root)))
```

代码运行截图 （至少包含有"Accepted"）

#44509087提交状态

查看 提交 统计

状态: Accepted

基本信息

#: 44509087
题目: 05455
提交人: 刘子暄
内存: 3664kB
时间: 28ms
语言: Python3
提交时间: 2024-04-02 19:51:31

源代码

```
class TreeNode:
    def __init__(self,val):
        self.val = val
        self.left = None
        self.right = None

def build_tree_insert(node,other_node_val):
    if node is None:
        return TreeNode(other_node_val)#None是节点的边界条件
    if other_node_val < node.val:
        node.left = build_tree_insert(node.left, other_node_val)
    elif other_node_val > node.val:
        node.right = build_tree_insert(node.right, other_node_val)
    return node#直观解释就是将现在的root与每一个node比较，判断放入做自主还是右子

def level_order_traversal(root):
    queue = [root]
    traversal = []
    while queue:
        node = queue.pop(0)
        traversal.append(str(node.val))
        if node.left:
            queue.append(node.left)
        if node.right:
            queue.append(node.right)
    return traversal

n = list(map(int, input().strip().split()))
nodes = list(dict.fromkeys(n))#使用set不稳定
root = None
for i in nodes:
    root = build_tree_insert(root, i)
print(' '.join(level_order_traversal(root)))
```

04078: 实现堆结构

http://cs101.openjudge.cn/practice/04078/

练习自己写个BinHeap。当然机考时候，如果遇到这样题目，直接import heapq。手搓栈、队列、堆、AVL等，考试前需要搓个遍。

思路：

代码

```
class BinHeap:
    def __init__(self):
        self.heapList = [0]
        self.currentSize = 0

    def precUp(self,i):
        while i//2 > 0:
            if self.heapList[i] < self.heapList[i//2]:
                tmp = self.heapList[i]
                self.heapList[i] = self.heapList[i//2]
                self.heapList[i // 2] = tmp
            i = i//2

    def insert(self,k):
        self.heapList.append(k)
        self.currentSize += 1
        self.precUp(self.currentSize)

    def precDown(self,i):
        while (i * 2) <= self.currentSize:
            minc = self.minChild(i)
            if self.heapList[minc] < self.heapList[i]:
                tmp = self.heapList[minc]
                self.heapList[minc] = self.heapList[i]
                self.heapList[i] = tmp
            i = minc

    def minChild(self, i):
        if i * 2 + 1 > self.currentSize:
            return i * 2
        else:
            if self.heapList[i * 2] < self.heapList[i * 2 + 1]:
                return i * 2
            else:
                return i * 2 + 1
    def delMin(self):
        popout = self.heapList[1]
        self.heapList[1] = self.heapList[self.currentSize]
        self.currentSize -= 1
```

```
        self.heapList.pop()
        self.precDown(1)
        return popout

BinHeap1 = BinHeap()

n = int(input())
for _ in range(n):
    a = list(map(int,input().split()))
    if a[0] == 1:
        BinHeap1.insert(a[1])
    elif a[0] == 2:
        print(BinHeap1.delMin())
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

状态: Accepted

源代码

```
class BinHeap:
    def __init__(self):
        self.heapList = [0]
        self.currentSize = 0

    def precUp(self,i):
        while i//2 > 0:
            if self.heapList[i] < self.heapList[i//2]:
                tmp = self.heapList[i]
                self.heapList[i] = self.heapList[i//2]
                self.heapList[i // 2] = tmp
            i = i//2

    def insert(self,k):
        self.heapList.append(k)
        self.currentSize += 1
        self.precUp(self.currentSize)

    def precDown(self,i):
        while (i * 2) <= self.currentSize:
            minc = self.minChild(i)
            if self.heapList[minc] < self.heapList[i]:
                tmp = self.heapList[minc]
                self.heapList[minc] = self.heapList[i]
                self.heapList[i] = tmp
            i = minc

    def minChild(self, i):
        if i * 2 + 1 > self.currentSize:
            return i * 2
        else:
            if self.heapList[i * 2] < self.heapList[i * 2 + 1]:
                return i * 2
            else:
                return i * 2 + 1
    def delMin(self):
        popout = self.heapList[1]
        self.heapList[1] = self.heapList[self.currentSize]
        self.currentSize -= 1
        self.heapList.pop()
        self.precDown(1)
        return popout

BinHeap1 = BinHeap()
```

基本信息

#: 44510988
题目: 04078
提交人: 刘子暄
内存: 4112kB
时间: 677ms
语言: Python3
提交时间: 2024-04-02 21:58:43

22161: 哈夫曼编码树

http://cs101.openjudge.cn/practice/22161/

思路：huffman手搓还是太痛苦了，看懂之后复制了数据处理的部分

代码

```
import heapq

class Node:
    def __init__(self, weight, char=None):
        self.weight = weight
        self.char = char
        self.left = None
        self.right = None

    def __lt__(self, other):
        if self.weight == other.weight:
            return self.char < other.char
        return self.weight < other.weight

def build_huffman_tree(characters):
    heap = []
    for char, weight in characters.items():
        heapq.heappush(heap, Node(weight, char))

    while len(heap) > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)
        merged = Node(left.weight + right.weight, min(left.char, right.char))
        merged.left = left
        merged.right = right
        heapq.heappush(heap, merged)

    return heap[0]

def encode_huffman_tree(root):
    codes = {}

    def traverse(node, code):
        if node.left is None and node.right is None:
            codes[node.char] = code
        else:
            traverse(node.left, code + '0')
            traverse(node.right, code + '1')
```

```
    traverse(root, '')
    return codes

def huffman_encoding(codes, string):
    encoded = ''
    for char in string:
        encoded += codes[char]
    return encoded

def huffman_decoding(root, encoded_string):
    decoded = ''
    node = root
    for bit in encoded_string:
        if bit == '0':
            node = node.left
        else:
            node = node.right

        if node.left is None and node.right is None:
            decoded += node.char
            node = root
    return decoded

n = int(input())
characters = {}
for _ in range(n):
    char, weight = input().split()
    characters[char] = int(weight)

huffman_tree = build_huffman_tree(characters)

codes = encode_huffman_tree(huffman_tree)

strings = []
while True:
    try:
        line = input()
        strings.append(line)
```

```
    except EOFError:
        break

results = []
for string in strings:
    if string[0] in ('0','1'):
        results.append(huffman_decoding(huffman_tree, string))
    else:
        results.append(huffman_encoding(codes, string))

for result in results:
    print(result)
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

状态：Accepted

源代码

```
import heapq

class Node:
    def __init__(self, weight, char=None):
        self.weight = weight
        self.char = char
        self.left = None
        self.right = None

    def __lt__(self, other):
        if self.weight == other.weight:
            return self.char < other.char
        return self.weight < other.weight

def build_huffman_tree(characters):
    heap = []
    for char, weight in characters.items():
        heapq.heappush(heap, Node(weight, char))

    while len(heap) > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)
        merged = Node(left.weight + right.weight, min(left.char, right.c
        merged.left = left
        merged.right = right
        heapq.heappush(heap, merged)

    return heap[0]

def encode_huffman_tree(root):
    codes = {}

    def traverse(node, code):
        if node.left is None and node.right is None:
            codes[node.char] = code
        else:
            traverse(node.left, code + '0')
            traverse(node.right, code + '1')

    traverse(root, '')
    return codes

def huffman_encoding(codes, string):
    encoded = ''
    for char in string:
```

基本信息

#:	44511306
题目:	22161
提交人:	刘子暄
内存:	3680kB
时间:	25ms
语言:	Python3
提交时间:	2024-04-02 22:27:41

晴问9.5: 平衡二叉树的建立

https://sunnywhy.com/sfbj/9/5/359

思路：avl就感觉是树的集大成者，出现的很多东西都使用了

代码

```python
class Node:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None
        self.height = 1

class AVL:
    def __init__(self):
        self.root = None

    def insert(self, value):
        if not self.root:
            self.root = Node(value)
        else:
            self.root = self._insert(value, self.root)

    def _insert(self, value, node):
        if not node:
            return Node(value)
        elif value < node.value:
            node.left = self._insert(value, node.left)
        else:
            node.right = self._insert(value, node.right)

        node.height = 1 + max(self._get_height(node.left), self._get_height(node.right))

        balance = self._get_balance(node)

        if balance > 1:
            if value < node.left.value:        # 树形是 LL
                return self._rotate_right(node)
            else:        # 树形是 LR
                node.left = self._rotate_left(node.left)
                return self._rotate_right(node)

        if balance < -1:
            if value > node.right.value:        # 树形是 RR
                return self._rotate_left(node)
```

```
            else:        # 树形是 RL
                node.right = self._rotate_right(node.right)
                return self._rotate_left(node)

        return node

    def _get_height(self, node):
        if not node:
            return 0
        return node.height

    def _get_balance(self, node):
        if not node:
            return 0
        return self._get_height(node.left) - self._get_height(node.right)

    def _rotate_left(self, z):
        y = z.right
        T2 = y.left
        y.left = z
        z.right = T2
        z.height = 1 + max(self._get_height(z.left), self._get_height(z.right))
        y.height = 1 + max(self._get_height(y.left), self._get_height(y.right))
        return y

    def _rotate_right(self, y):
        x = y.left
        T2 = x.right
        x.right = y
        y.left = T2
        y.height = 1 + max(self._get_height(y.left), self._get_height(y.right))
        x.height = 1 + max(self._get_height(x.left), self._get_height(x.right))
        return x

    def preorder(self):
        return self._preorder(self.root)

    def _preorder(self, node):
        if not node:
```

```
            return []
        return [node.value] + self._preorder(node.left) + self._preorder(node.right)

n = int(input().strip())
sequence = list(map(int, input().strip().split()))

avl = AVL()
for value in sequence:
    avl.insert(value)

print(' '.join(map(str, avl.preorder())))
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

02524: 宗教信仰

http://cs101.openjudge.cn/practice/02524/

思路：没太看懂思路，之后在好好看一下

代码

```python
def init_set(n):
    return list(range(n))

def get_father(x, father):
    if father[x] != x:
        father[x] = get_father(father[x], father)
    return father[x]

def join(x, y, father):
    fx = get_father(x, father)
    fy = get_father(y, father)
    if fx == fy:
        return
    father[fx] = fy

def is_same(x, y, father):
    return get_father(x, father) == get_father(y, father)

def main():
    case_num = 0
    while True:
        n, m = map(int, input().split())
        if n == 0 and m == 0:
            break
        count = 0
        father = init_set(n)
        for _ in range(m):
            s1, s2 = map(int, input().split())
            join(s1 - 1, s2 - 1, father)
        for i in range(n):
            if father[i] == i:
                count += 1
        case_num += 1
        print(f"Case {case_num}: {count}")

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

代码运行截图 （AC代码截图，至少包含有"Accepted"）

#44511871提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
def init_set(n):
    return list(range(n))

def get_father(x, father):
    if father[x] != x:
        father[x] = get_father(father[x], father)
    return father[x]

def join(x, y, father):
    fx = get_father(x, father)
    fy = get_father(y, father)
    if fx == fy:
        return
    father[fx] = fy

def is_same(x, y, father):
    return get_father(x, father) == get_father(y, father)

def main():
    case_num = 0
    while True:
        n, m = map(int, input().split())
        if n == 0 and m == 0:
            break
        count = 0
        father = init_set(n)
        for _ in range(m):
            s1, s2 = map(int, input().split())
            join(s1 - 1, s2 - 1, father)
        for i in range(n):
            if father[i] == i:
                count += 1
        case_num += 1
        print(f"Case {case_num}: {count}")

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

基本信息

#: 44511871
题目: 02524
提交人: 刘子暄
内存: 5844kB
时间: 1243ms
语言: Python3
提交时间: 2024-04-02 23:21:00

©2002-2022 POJ 京ICP备20010980号-1 English 帮助 关于

2. 学习总结和收获

如果作业题目简单，有否额外练习题目，比如：OJ“2024spring每日选做”、CF、LeetCode、洛谷等网站题目。

突然发现树如其名，“树”都是从“根”长出来的，所以大部分建树最后给出的都是root，而root就包含了树的全部信息，同样的，建树过程是一个叶子一个叶子去长出来的，所以建树过程中树就在不断更新，不论是直接建树还是插入节点建树都是一样的，只是更新操作有的放在递归中，有的直接用函数去做了

二叉搜索树建树类型：

给出数据为前序遍历形式，root为头，确定root后从前向后进行比较，将数据分为左子树（小于根节点）和右子树（大于根节点）并递归执行建树操作,也就是说前序遍历中，所有小于root的值都在左边，大于root的都在右边
给出数据为后序遍历，root为尾，确定后遍历方式相同
给出数据为中序遍历，暂不会
给出数据无序，用插入来建树，因为无序所以每个都要判断并重新插入
（代码都在上面）

二叉搜索树输出格式函数;
层次遍历：建缓冲栈，将每层节点数据压入缓冲栈后再下沉至下一层

把AVL照抄了一遍，理解没问题，但是各个模块之间的联系有点模糊，互相调用很频繁，需要再看一下