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

Updated 2214 GMT+8 March 31, 2024

2024 spring, Complied by ==张宇帆 心理与认知科学学院==

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）==

Python编程环境：Spyder IDE 5.2.2

1. 题目

22275: 二叉搜索树的遍历

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

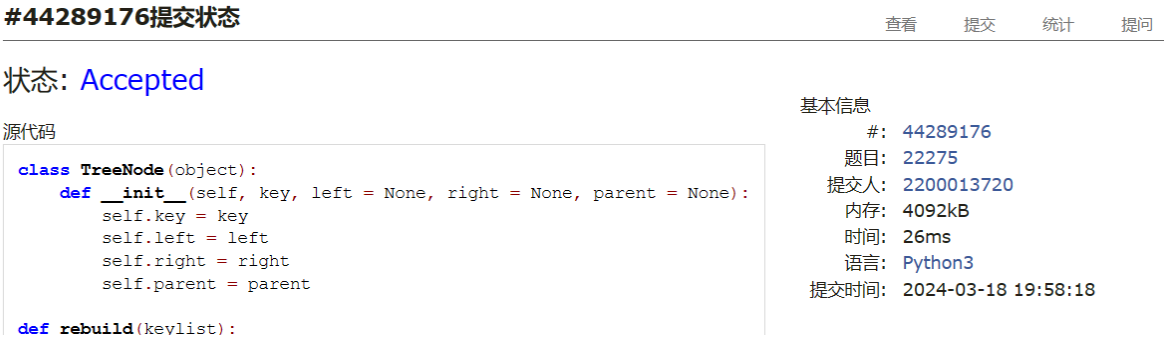
思路：依据前序遍历的特性构建二叉搜索树，然后进行后序遍历即可

代码

```
#
class TreeNode(object):
    def __init__(self, key, left = None, right = None, parent = None):
        self.key = key
        self.left = left
        self.right = right
        self.parent = parent

def rebuild(keylist):
    if len(keylist) == 1:
        return TreeNode(keylist[0])
    root = keylist[0]
```

```
    idx = 1
    for i in range(1, len(keylist)):
        if keylist[i] > root:
            idx = i
            break
    temp = TreeNode(root)
    if not len(keylist[1:idx]) == 0:
        temp.left = rebuild(keylist[1:idx])
    if not len(keylist[idx:]) == 0:
        temp.right = rebuild(keylist[idx:])
    return temp

def postloop(tree, result):
    if not tree == None:
        result = postloop(tree.left, result)
        result = postloop(tree.right, result)
        result.append(str(tree.key))
    return result


n = int(input())
numlist = list(map(int,input().split()))
tree = rebuild(numlist)
result = []
print(' '.join(postloop(tree, result)))
```

代码运行截图 ==（至少包含有"Accepted"）==



05455: 二叉搜索树的层次遍历

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

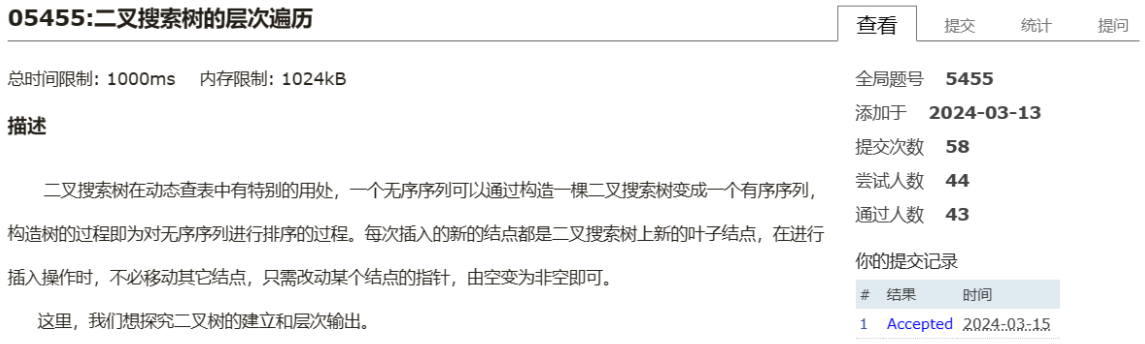
思路：维护一个二叉搜索树的列表即可，然后按照层次遍历输出

代码

```
#
class TreeNode(object):
    def __init__(self, root, left = None, right = None, parent = None):
        self.root = root
        self.left = left
```

```
        self.right = right
        self.parent = parent

def layerloop(treenode, result = []):
    queue = [treenode]
    while queue:
        for _ in range(len(queue)):
            node = queue.pop(0)
            result.append(str(node.root))
            if node.left:
                queue.append(node.left)
            if node.right:
                queue.append(node.right)
    return ' '.join(result)

numlist = list(map(int,input().split()))
n = len(numlist)
trees = [TreeNode(i) for i in range(n)]
for i in range(n):
    trees[i].root = numlist[i]
    temp = trees[0]
    while temp:
        if temp.root == numlist[i]:
            break
        elif temp.root < numlist[i] and not temp.right == None:
            temp = temp.right
        elif temp.root > numlist[i] and not temp.left == None:
            temp = temp.left
        elif temp.root < numlist[i]:
            temp.right = trees[i]
            break
        elif temp.root > numlist[i]:
            temp.left = trees[i]
            break
print(layerloop(trees[0]))
```

代码运行截图 ==（至少包含有"Accepted"）==



05455:二叉搜索树的层次遍历

查看 提交 统计 提问

总时间限制: 1000ms 内存限制: 1024kB

描述

二叉搜索树在动态查表中有特别的用处，一个无序序列可以通过构造一棵二叉搜索树变成一个有序序列，构造树的过程即为对无序序列进行排序的过程。每次插入的新的结点都是二叉搜索树上新的叶子结点，在进行插入操作时，不必移动其它结点，只需改动某个结点的指针，由空变为非空即可。

这里，我们想探究二叉树的建立和层次输出。

全局题号 **5455**
添加于 **2024-03-13**
提交次数 **58**
尝试人数 **44**
通过人数 **43**

你的提交记录

#	结果	时间
1	Accepted	2024-03-15

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：在用heapq做完后就在想自己手搓一个，也算是debug了一会才实现，感觉最重要的就是上浮下浮两个动作，写起来简单理解起来有点难度。

代码

```
#
## 堆的列表实现
class heap_list(object):
    def __init__(self, List):
        self.heap = sorted(List)
        self.length = len(List)

    def getidx(self, relation, idx): ## 获取列表第idx位元素的父节点和左右子节点的索引
        if relation == 'parent':
            return (idx-1)//2
        elif relation == 'left':
            return 2*idx + 1
        elif relation == 'right':
            return 2*idx + 2

    def minchild(self, idx): ## 返回值最小的子节点，如果没有则返回None
        if 2*idx + 1 >= self.length:
            return None
        elif 2*idx + 2 >= self.length:
            return 2*idx + 1
        else:
            return [2*idx + 1, 2*idx + 2][self.heap[2*idx + 1] > self.heap[2*idx
+ 2]]

    def up(self, idx): ## 将列表第idx位元素进行上浮
        while idx > 0 and self.heap[idx] < self.heap[self.getidx('parent', idx)]:
            parent_idx = self.getidx('parent', idx)
            self.heap[idx], self.heap[parent_idx] = self.heap[parent_idx],
self.heap[idx]
            idx = parent_idx

    def down(self, idx): ## 将列表第idx位元素进行下沉
        child = self.minchild(idx)
        if not child == None:
            if self.heap[child] < self.heap[idx]:
                self.heap[idx], self.heap[child] = self.heap[child],
self.heap[idx]
            if not idx == child:
                self.down(child)


    def popmin(self): ## 弹出列表最小值
```

```
        if self.length == 0:
            return None
        else:
            Min = self.heap[0]
            self.length -= 1
            self.heap[0] = self.heap[-1]
            self.heap.pop()
            self.down(0)
            return Min

    def push(self, val): ## 添加新数据
        self.heap.append(val)
        self.length += 1
        self.up(self.length-1)

n = int(input())
K = heap_list([])
for _ in range(n):
    order = input()
    if order[0] == '1':
        K.push(int(order[2:]))
    elif order[0] == '2':
        print(K.popmin())
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

22161: 哈夫曼编码树

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

思路：一开始想着既要用堆去取最小的两个权重，又要建树进行编码，有点“知难而退，要不直接看答案？”的想法，后来想着不要害怕不要嫌麻烦，得自己写一遍。然后琢磨了半个多小时才搞明白思路自己写了出来，很高兴一遍AC了，只不过不知道标准答案是怎么样的，感觉自己写得繁琐了些。
具体思路就还是用堆取最小的两个权重然后构建节点，最后形成一颗完整的树。为了避免之后还要遍历求叶节点，在构建树的时候把编码也给加上了，因此最后输出的tree就包含了：该树包含的字典集，各个字符对应的编码，两个列表在下标上保持一一对应的关系。这么看来这也是有递归的思想在里面。

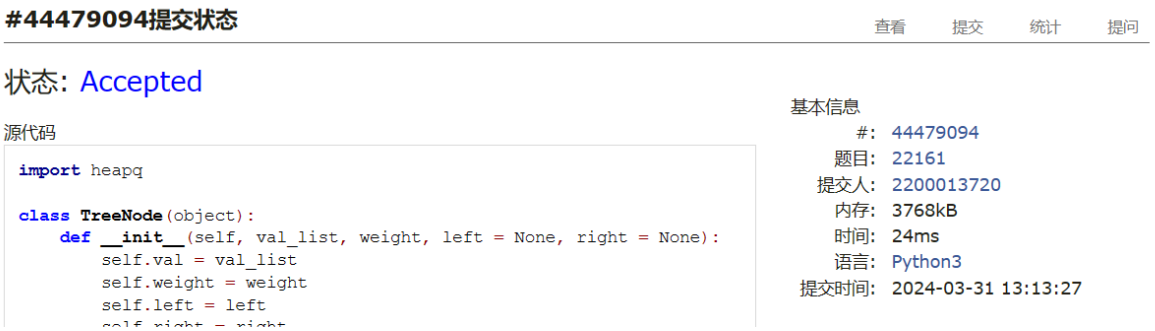
代码

```
```

```
#
import heapq

class TreeNode(object):
    def __init__(self, val_list, weight, left = None, right = None):
        self.val = val_list
        self.weight = weight
        self.left = left
        self.right = right
        self.code = []

    def isleaf(self):
        return self.left == None and self.right == None

n = int(input())
codes = []
for _ in range(n):
    code = input().split()
    code, weight = code[0], int(code[1])
    temp = TreeNode([code], weight)
    codes.append([weight, ord(code), temp])
heapq.heapify(codes)
for i in range(n-1):
    left = heapq.heappop(codes)
    right = heapq.heappop(codes)
    neword = min(left[1], right[1])
    newweight = left[0]+right[0]
    newtemp = TreeNode(left[2].val+right[2].val, newweight)
    if left[2].code == []:
        newtemp.code.append('0')
    else:
        for idx in left[2].code:
            newtemp.code.append('0'+idx)
    if right[2].code == []:
        newtemp.code.append('1')
    else:
        for idx in right[2].code:
            newtemp.code.append('1'+idx)
    newtemp.left = left[2]
    newtemp.right = right[2]
    heapq.heappush(codes, [newweight, neword, newtemp])
tree = codes[0][2]
codes = {}
recodes = {}
for i in range(n):
    codes[tree.val[i]] = tree.code[i]
    recodes[tree.code[i]] = tree.val[i]

while True:
    try:
        order = input()
        result = ''
        o = ''
        if '0' in order or '1' in order:
            for idx in range(len(order)):
                o += order[idx]
```

```
                if o in recodes:
                    result += recodes[o]
                    o = ''
        else:
            for idx in range(len(order)):
                o += order[idx]
                if o in codes:
                    result += codes[o]
                    o = ''
        print(result)
    except EOFError:
        break
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==



晴问9.5: 平衡二叉树的建立

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

思路：手搓的时候真的很痛苦，按照之前的印象写了一小时多还是无法实现，最后实在没办法去看了作业题解，结果发现关键部分的实现不太一样，作业题解是用height作为节点属性，而我一开始想的是balance，感觉我一开始的思路也没什么问题，但就是实现不了，之后还得找时间再好好看看。

代码

```
#
class TreeNode(object):
    def __init__(self, val):
        self.root = val
        self.left = None
        self.right = None
        self.parent = None
        self.height = 1

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

    def leftrotate(self, node):
        newroot = node.right
        newchild = newroot.left
```

```
        newroot.left = node
        node.right = newchild
        node.height = 1 + max(self.get_height(node.left),
self.get_height(node.right))
        newroot.height = 1 + max(self.get_height(newroot.left),
self.get_height(newroot.right))
        return newroot

    def rightrotate(self, node):
        newroot = node.left
        newchild = newroot.right
        newroot.right = node
        node.left = newchild
        node.height = 1 + max(self.get_height(node.left),
self.get_height(node.right))
        newroot.height = 1 + max(self.get_height(newroot.left),
self.get_height(newroot.right))
        return newroot

    def push(self, val):
        if not self.root:
            self.root = TreeNode(val)
        else:
            self.root = self._push(val, self.root)

    def _push(self, val, currentnode):
        if not currentnode:
            return TreeNode(val)
        elif val < currentnode.root:
            currentnode.left = self._push(val, currentnode.left)
        else:
            currentnode.right = self._push(val, currentnode.right)
        currentnode.height = 1 + max(self.get_height(currentnode.left),
self.get_height(currentnode.right))

        balance = self.get_balance(currentnode)
        if balance > 1:
            if val < currentnode.left.root:
                return self.rightrotate(currentnode)
            else:
                currentnode.left = self.leftrotate(currentnode.left)
                return self.rightrotate(currentnode)

        if balance < -1:
            if val > currentnode.right.root:
                return self.leftrotate(currentnode)
            else:
                currentnode.right = self.rightrotate(currentnode.right)
                return self.leftrotate(currentnode)

        return currentnode

    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 preloop(tree, result):
    if not tree == None:
        result.append(str(tree.root))
        result = preloop(tree.left, result)
        result = preloop(tree.right, result)
    return result

n = int(input())
numlist = list(map(int,input().split()))
tree = AVLtree()
for idx in range(n):
    tree.push(numlist[idx])
print(' '.join(preloop(tree.root, [])))
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

提交时间	结果	时长(ms)	语言	
2024-03-31 16:03:25	完美通过	0	Python	查看

02524: 宗教信仰

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

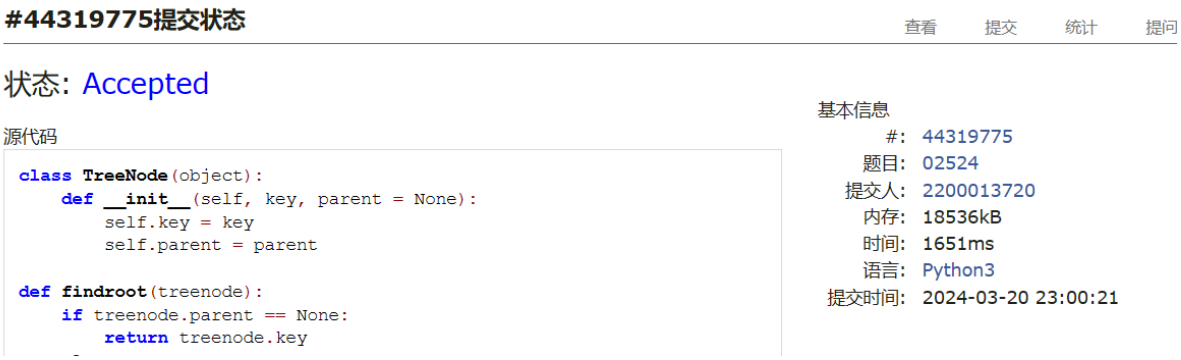
思路：最开始接触到这道题就想着用列表的值去表示，但是一直担心TLE于是没有考虑到“取根值“，导致了总是WA，后来想了想用树做，也是没有考虑到父节点比当前节点大等情况，后来想通了直接比较根节点就AC了，然后才了解到并查集这个概念，只能安慰自己好歹用树写了一个）

代码

```
#
class TreeNode(object):
    def __init__(self, key, parent = None):
        self.key = key
        self.parent = parent

def findroot(treenode):
    if treenode.parent == None:
        return treenode.key
    else:
        return findroot(treenode.parent)
```

```
n,m = map(int,input().split())
count = 0
while not n == 0 and not m == 0:
    count += 1
    trees = [TreeNode(i) for i in range(n)]
    result = 0
    for _ in range(m):
        i,j = map(int,input().split())
        i,j = findroot(trees[i-1]), findroot(trees[j-1])
        if j < i:
            trees[i].parent = trees[j]
        elif j > i:
            trees[j].parent = trees[i]
    for i in range(n):
        if trees[i].parent == None:
            result += 1
    print('Case %s: %s'%(count, result))
    n,m = map(int,input().split())
```

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

==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。==

这一周由于任务量多起来了，而且刚好大部分课程的任务都集中在了这个周末，所以也没有好好地落实预定计划，只能在每天早上起来上课之前把当天的每日选做两道题给完成了，再加上AVL和哈夫曼编码让我感到有点头疼（哈夫曼做出来后感觉还好，AVL树感觉还是懵懵的），导致这次作业这么晚交。接下来要好好提高自己的效率，再重新好好分配各个课程的时间。