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

Updated 2214 GMT+8 March 24, 2024

2024 spring, Complied by 夏天，生命科学学院

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）

操作系统：Windows 10 家庭版

Python编程环境：Spyder python(3.11)

1. 题目

22275: 二叉搜索树的遍历

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

思路：前序表达式的第一个为根节点，小于该根节点的值的部分为左子树的前序表达式，大于根节点的值的部分为右子树的前序表达式，然后递归即可

代码

```
    def prefix_to_postfix(prefix):
        if not prefix:
            return []
        root=prefix[0]
        left_prefix=[i for i in prefix if i<root]
        right_prefix=[j for j in prefix if j>root]
        return prefix_to_postfix(left_prefix)+prefix_to_
    postfix(right_prefix)+[root]
    n=int(input())
    prefix=list(map(int,input().split()))
    postfix=prefix_to_postfix(prefix)
    print(' '.join(map(str,postfix)))
```

代码运行截图 （至少包含有"Accepted"）

状态: Accepted

源代码

```
def prefix_to_postfix(prefix):
    if not prefix:
        return []
    root=prefix[0]
    left_prefix=[i for i in prefix if i<root]
    right_prefix=[j for j in prefix if j>root]
    return prefix_to_postfix(left_prefix)+prefix_to_postfix(right_prefi
n=int(input())
prefix=list(map(int,input().split()))
postfix=prefix_to_postfix(prefix)
print(' '.join(map(str,postfix)))
```

基本信息

#:	44407437
题目:	22275
提交人:	23n2300012289
内存:	3864kB
时间:	26ms
语言:	Python3
提交时间:	2024-03-26 15:37:14

05455: 二叉搜索树的层次遍历

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

思路：逐个插入，比当前节点大的向右走；比当前节点小的向左走

代码

```
from collections import deque
class Treenode:
    def __init__(self,val):
        self.val=val
        self.left=None
        self.right=None
def insert(root,key):
    if not root:
        return Treenode(key)
    if root.val>key:
        root.left=insert(root.left,key)
    if root.val<key:
        root.right=insert(root.right,key)
    return root
def level_order_traversal(root):
    result=[]
    queue=deque([root])
    while queue:
        node=queue.popleft()
        result.append(node.val)
        if node.left:
            queue.append(node.left)
        if node.right:
            queue.append(node.right)
    return result
root=None
keys=list(map(int,input().split()))
for key in keys:
    root=insert(root,key)
print(' '.join(map(str,level_order_traversal(root))))
```

代码运行截图 （至少包含有"Accepted"）

04078: 实现堆结构

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

思路：见注释

代码

```
class Binheap:
    def __init__(self):
        self.heaplist=[0]
        self.currentsize=0
    def percup(self,i): #向上交换
        while i//2>0: #注意到索引为i的节点，其父节点为i//2
            if self.heaplist[i]<self.heaplist[i//2]: #小于则交换
                tmp=self.heaplist[i]
                self.heaplist[i]=self.heaplist[i//2]
                self.heaplist[i//2]=tmp
            i//=2
    def insert(self,k): #将元素放入“堆”中，然后进行向上交换使堆顶元素最小
        self.heaplist.append(k)
        self.currentsize+=1
        self.percup(self.currentsize)
    def percdown(self,i): #向下交换
        while i*2<=self.currentsize: #当前节点有子节点
            mc=self.minchild(i)
            if self.heaplist[i]>self.heaplist[mc]:
                tmp=self.heaplist[i]
                self.heaplist[i]=self.heaplist[mc]
                self.heaplist[mc]=tmp
            i=mc
    def minchild(self,i): #获得较小子节点的索引
        if i*2+1>self.currentsize: #当前节点只有一个子节点
            return i*2
        if self.heaplist[2*i]<self.heaplist[2*i+1]: #当前节点有两个子节点，比较得到较小的那个
            return i*2
        else:
            return i*2+1
    def delmin(self): #弹出堆顶元素，然后将最后一个入堆的元素放到堆顶（保证堆的结构性质）并进行向下交换
        tmp=self.heaplist[1]
        self.heaplist[1]=self.heaplist[self.currentsize]
        self.currentsize-=1
        self.heaplist.pop()
        self.percdown(1)
        return tmp
n=int(input())
bh=Binheap()
for _ in range(n):
   input_=input().split()
   if input_[0]=='1':
        bh.insert(int(input_[1]))
   else:
       print(str(bh.delmin()))
```

代码运行截图

（AC代码截图，至少包含有"Accepted"）

22161: 哈夫曼编码树

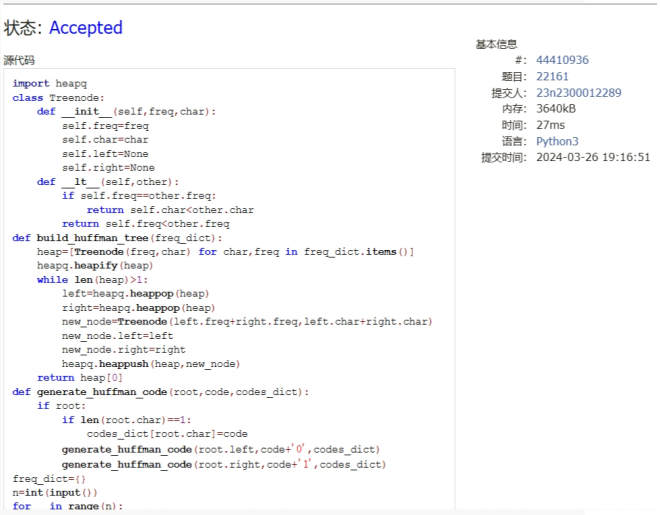
http://cs101.openjudge.cn/practice/22161/

思路：跟据定义建树即可，注意只有char的长度为1是才能入字典

代码

```
import heapq
class Treenode:
    def __init__(self,freq,char):
        self.freq=freq
        self.char=char
        self.left=None
        self.right=None
    def __lt__(self,other):
        if self.freq==other.freq:
            return self.char<other.char
        return self.freq<other.freq
def build_huffman_tree(freq_dict):
    heap=[Treenode(freq,char) for char,freq in freq_dict.items()]
    heapq.heapify(heap)
    while len(heap)>1:
        left=heapq.heappop(heap)
        right=heapq.heappop(heap)
        new_node=Treenode(left.freq+right.freq,left.char+right.char)
        new_node.left=left
        new_node.right=right
        heapq.heappush(heap,new_node)
    return heap[0]
def generate_huffman_code(root,code,codes_dict):
    if root:
        if len(root.char)==1:
            codes_dict[root.char]=code
        generate_huffman_code(root.left,code+'0',codes_dict)
        generate_huffman_code(root.right,code+'1',codes_dict)
freq_dict={}
n=int(input())
for _ in range(n):
    char,freq=input().split()
    freq_dict[char]=int(freq)
root=build_huffman_tree(freq_dict)
codes_dict={}
generate_huffman_code(root,'',codes_dict)
while True:
    try:
        s=input()
        if s.isdigit():
            decode=''
            current=''
            for bit in s:
                current+=bit
                if current in codes_dict.values():
                    decode+=list(codes_dict.keys())[list(codes_dict.values()).index(current)]
                    current=''
            print(decode)
        else:
            encode=[codes_dict[bit] for bit in s]
            print(''.join(encode))
    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：见注释 代码

```
class Treenode:
    def __init__(self,val):
        self.val=val
        self.left=None
        self.right=None
        self.height=1
class AVLtree:
    def get_height(self,node): #获取树的高度
        if not node:
            return 0
        return node.height
    def get_balance_factor(self,node): #获取平衡因子
        if not node:
            return 0
        return self.get_height(node.left)-self.get_height(node.right)
    def right_rotate(self,lost_balance_node): #右旋操作
        new_node=lost_balance_node.left #失衡节点的左子节点成为新节点
        right_subtree=new_node.right #新节点的原右子树
        new_node.right=lost_balance_node #失衡节点作为新节点的右子节点
        lost_balance_node.left=right_subtree #新节点的右子树成为失衡节点的左子树
        lost_balance_node.height=1+max(self.get_height(lost_balance_node.left),self.get_height(lost_balance_node.right))
        new_node.height=1+max(self.get_height(new_node.left),self.get_height(new_node.right)) #更新高度
        return new_node
    def left_rotate(self,lost_balance_node): #左旋操作，与右旋相反
        new_node=lost_balance_node.right
        left_subtree=new_node.left
        new_node.left=lost_balance_node
        lost_balance_node.right=left_subtree
        lost_balance_node.height=1+max(self.get_height(lost_balance_node.left),self.get_height(lost_balance_node.right))
        new_node.height=1+max(self.get_height(new_node.left),self.get_height(new_node.right))
        return new_node
```

```
def insert(self,node,key): #插入AVL树
        if not node:
            return Treenode(key)
        if node.val>key:
            node.left=self.insert(node.left,key)
        if node.val<key:
            node.right=self.insert(node.right,key)
        node.height=1+max(self.get_height(node.left),self.get_height(node.right))
        balance_factor=self.get_balance_factor(node)
        if balance_factor>1 and key<node.left.val: #LL型失衡，右旋一次
            return self.right_rotate(node)
        if balance_factor<-1 and key>node.right.val: #RR型失衡，左旋一次
            return self.left_rotate(node)
        if balance_factor>1 and key>node.left.val: #LR型失衡，失衡节点的左子树左旋一次，然后整个树右旋一次
            node.left=self.left_rotate(node.left)
            return self.right_rotate(node)
        if balance_factor<-1 and key<node.right.val: #RL型失衡，失衡节点右子树右旋一次，然后整个树左旋一次
            node.right=self.right_rotate(node.right)
            return self.left_rotate(node)
        return node
def preorder(node):
    if not node:
        return []
    res=[node.val]
    res+=preorder(node.left)
    res+=preorder(node.right)
    return res
avltree=AVLtree()
root=None
n=int(input())
keys=list(map(int,input().split()))
for key in keys:
    root=avltree.insert(root,key)
print(' '.join(map(str,preorder(root))))
```

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：见注释

代码

```
def find(i): #寻找根节点
    if parent[i]!=i: #如果当前节点的根节点不是自己
        parent[i]=find(parent[i])
    return parent[i]
def union(x,y): #使y的根节点成为x的根节点的根节点
    parent[find(x)]=find(y)
i=0
while True:
    n,m=map(int,input().split())
    if n==m==0:
        break
    i+=1
    parent=list(range(n+1)) #初始化每个结点的根节点为自己
    for _ in range(m):
        a,b=map(int,input().split())
        union(a,b)
    religions=set(find(x) for x in range(1,n+1))
    print('Case '+format(i)+': '+format(len(religions)))
```

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。

这次作业挺难的，不过题越难收获越多：①在看完课上讲的binheap的代码后自己能完整的手搓出来；②对于二叉搜索树和平衡二叉树有初步了解（平衡二叉树的四种失衡类型及对应使其恢复平衡的旋转方法是比较重要的概念）；③并查集思想最精妙之处——“你只看见我渺小的身躯，却没有看到我心中的广阔森林。”（摘自知乎）