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

Updated 2214 GMT+8 March 24, 2024

2024 spring, Complied by 城环 吴至超

**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：Windows11

Python编程环境：pycharm2023.2.3

# 1. 题目

## 22275: 二叉搜索树的遍历

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

思路：

二叉搜索树，由于左<中<右，所以其前序遍历就是权值顺序，结合上次作业可以解出。

代码

```
# n=int(input())
preorder=[int(x) for x in input().split()]#前序遍历
class tree():
    def __init__(self,value):
        self.value=value
        self.left=None
        self.right=None

inorder=[int(x) for x in preorder]
inorder.sort()
```

```
def buildtree(preorder,inorder):
    if not preorder or not inorder:#边界条件，要么inorder一边已经建立完，要么建立到最右边
preorder已经为空
        return None
    else:

        a=inorder.index(preorder[0])
        treenode=tree(preorder.pop(0))
        treenode.left=buildtree(preorder,inorder[:a])
        treenode.right=buildtree(preorder,inorder[a+1::])
    return treenode

def postorder(root):
    stack=[]
    if root:
        stack.extend(postorder(root.left))
        stack.extend(postorder(root.right))
        stack.append(root.value)
    return stack

print(" ".join(map(str,postorder(buildtree(preorder,inorder)))))
```

代码运行截图 ==（至少包含有"Accepted"）==

题目 排名 状态 提问

**#44445483提交状态**

查看 提交 统计 提问

状态: Accepted

源代码

```
n=int(input())
preorder=[int(x) for x in input().split()]#前序遍历
class tree():
    def __init__(self,value):
        self.value=value
        self.left=None
        self.right=None

inorder=[int(x) for x in preorder]
inorder.sort()
def buildtree(preorder,inorder):
    if not preorder or not inorder:#边界条件，要么inorder一边已经建立完，要么
        return None
    else:

        a=inorder.index(preorder[0])
        treenode=tree(preorder.pop(0))
        treenode.left=buildtree(preorder,inorder[:a])
        treenode.right=buildtree(preorder,inorder[a+1::])
    return treenode

def postorder(root):
    stack=[]
    if root:
        stack.extend(postorder(root.left))
        stack.extend(postorder(root.right))
        stack.append(root.value)
    return stack

print(" ".join(map(str,postorder(buildtree(preorder,inorder)))))
```

基本信息

| | |
|---|---|
| #: | 44445483 |
| 题目: | 22275 |
| 提交人: | 23n2300013289 |
| 内存: | 3940kB |
| 时间: | 25ms |
| 语言: | Python3 |
| 提交时间: | 2024-03-29 15:55:23 |



English 帮助 关于

# 05455: 二叉搜索树的层次遍历

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

思路：a=list(dict.fromkeys(a))的方法对列表去重，建树递归的思路，从根节点开始，每次搭配一棵新树，比较与当前根节点的权值，考虑向左或者向右建树。

层次遍历就是在列表里不断扩展

代码

```
# #05455 二叉搜索树的层次遍历
class tree:
    def __init__(self,value):
        self.value=value
        self.left=None
        self.right=None
a=list(int(x) for x in input().split())#去重的操作！神奇
a=list(dict.fromkeys(a))

def buildtree(root,numm):#root表示来到的节点位置，numm表示新建立的树
    if root==None:#结束递归，来到空节点并赋值
        return numm
    else:
        if numm.value>root.value:#如果比当前节点的价值高，那么向右边推进比较
            root.right=buildtree(root.right,numm)
        else:#同理
            root.left=buildtree(root.left,numm)
    return root#想要每次从根节点开始遍历，并使得层次遍历时可以直接调用此函数

root=None#确保根节点可以在上述函数中被填入
for i in a:
    treenode=tree(i)
    root=buildtree(root,treenode)

def layer(root):#进行层次遍历
    stack=[root]
    for i in stack:
        if i.left!=None:
            stack.append(i.left)
        if i.right!=None:
            stack.append(i.right)
    pri=[str(x.value) for x in stack]

    return " ".join(pri)

print(layer(root))
```

代码运行截图 ==（至少包含有"Accepted"）==

#44447691提交状态

查看 提交 统计 提问

状态: Accepted

基本信息

#: 44447691
题目: 05455
提交人: 23n2300013289
内存: 4008kB
时间: 26ms
语言: Python3
提交时间: 2024-03-29 18:09:00

源代码

```
#05455 二叉搜索树的层次遍历
class tree:
    def __init__(self,value):
        self.value=value
        self.left=None
        self.right=None
a=list(int(x) for x in input().split())#去重的操作！神奇
a=list(dict.fromkeys(a))

def buildtree(root,numm):#root表示来到的节点位置，numm表示新建立的树
    if root==None:#结束递归，来到空节点并赋值
        return numm
    else:
        if numm.value>root.value:#如果比当前节点的价值高，那么向右边推进比较
            root.right=buildtree(root.right,numm)
        else:#同理
            root.left=buildtree(root.left,numm)
    return root#想要每次从根节点开始遍历，并使得层次遍历时可以直接调用此函数

root=None#确保根节点可以在上述函数中被填入
for i in a:
    treenode=tree(i)
    root=buildtree(root,treenode)

def layer(root):#进行层次遍历
    stack=[root]
    for i in stack:
        if i.left!=None:
            stack.append(i.left)
        if i.right!=None:
            stack.append(i.right)
    pri=[str(x.value) for x in stack]

    return " ".join(pri)
```

# 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：堆就是个列表，其序号由于开头的0使得各个根节点与左右子节点的序号存在简单的数学联系。

在构建时，不断在列表末尾加入新的子树，考虑到堆的有序性，加入之后利用perup函数不断与根节点交换，上移到合适的位置

在弹出最小节点后，为了堆的结构性，把末尾的节点加到堆顶来，在左右比较下沉，与左右节点中权值较小的点交换位置

代码

```
# class binheap:#binheap也就是一个列表
    def __init__(self):#内置的
        self.currentsize=0
        self.heap=[0]
    def perup(self,i):#i初始为当前currentsize，局部变量，与init中的无关
        while i//2>0: #i逐渐表示此时a来到的节点位置，i//2表示预备位置，i在1时就不用再比较了
            if self.heap[i]<self.heap[i//2]:
                tempo=self.heap[i//2]
                self.heap[i//2]=self.heap[i]
                self.heap[i]=tempo
            i=i//2
    def insert(self,a):#一种操作
```

```
        self.heap.append(a)#接在列表末端
        self.currentsize+=1#当前节点数，也就是列表长度减去1
        self.perup(self.currentsize)#上浮换位置,self.currentsize只作为该函数内的局部变
量

    #接下来还需要重建堆，因为根节点已经移除
    #仍然需要保持堆的结构性与有序性，
    #首先，结构性-->把尾巴的叶子结点移动到根节点
    #随后,有序性-->根节点下沉
    def perdown(self,i):#i表示根节点目前所居位置，即初始为1
        while i*2<=self.currentsize:#
            if i*2+1<=self.currentsize:
                if self.heap[i*2]<self.heap[i*2+1]:
                    idx=i*2
                    minn=self.heap[i*2]
                else:
                    idx=i*2+1
                    minn=self.heap[i*2+1]
                if self.heap[i]>minn:
                    self.heap[idx]=self.heap[i]
                    self.heap[i]=minn
                i=idx#及时更新
            else:#i*2+1已越界
                if self.heap[i]>self.heap[i*2]:
                    tempo=self.heap[i*2]
                    self.heap[i*2]=self.heap[i]
                    self.heap[i]=tempo
                i=i*2

    def reconstruct(self):
        self.heap.insert(1,self.heap.pop())
        self.perdown(1)

n=int(input())#操作次数
heap1=binheap()
for y in range(n):
    a=list(map(int,input().split()))
    if a[0]==1:
        heap1.insert(a[1])
    else:
        print(heap1.heap.pop(1))
        heap1.currentsize-=1
        heap1.reconstruct()
```

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：import堆来建立哈夫曼编码树，不断合并权值最小的2个节点，把他们分别作为新树的左右子树，新树的权值为两树之和，名称为None，直到列表长度为1，即所得哈夫曼编码树，按题目定义最小。

数字转字母：相对好写一些，正常地更新当前位置就好。

字母转数字：建立一个列表，把各个字母对应的01编码放进去。具体实现办法为递归，（当前根节点，当前节点开始的编码），类似这样左右的递归写法能够遍历到每一个字母。

代码

```
#
import heapq
class tree:
    def __init__(self,value,char):
        self.value=value
        self.left=None
        self.right=None
        self.char=char
    def __lt__(self,other):#用来判断哪个节点更小，self与other同是tree
        if self.value==other.value:
            return self.char<other.char
        else:
```

```
            return self.value<other.value

def buildtree(dict):#建树，合并
    heap=[]
    for char,value in dict.items():#建堆
        heapq.heappush(heap,tree(value,char))
    while len(heap)>1:
        left=heapq.heappop(heap)#
        right=heapq.heappop(heap)
        merged=tree(left.value+right.value,None)
        merged.left=left
        merged.right=right
        heapq.heappush(heap,merged)
    return heap[0]#根节点

def find(root):#难点,字母转数字
    codes = {}  # 用来储藏每个字符的对应编码
    def xunzhao(tree,code):
        if tree.left is None and tree.right is None:#到头了
            codes[tree.char]=code
        else:#实现：在向下深度递归的同时，把每一个节点对应编码也给储存在字典codes里
            xunzhao(tree.left,code+"0")
            xunzhao(tree.right,code+"1")
    xunzhao(root,"")
    return codes

def release(root,sample):#数字转字母,root堆根，sampel是str
    output=""
    tempo=root
    for k in sample:
        if k=="0" :
            if tempo.left:
                if tempo.left.char!=None:
                    output+=tempo.left.char
                    tempo=root
                else:
                    tempo=tempo.left
        else:
            if tempo.right:
                if tempo.right.char!=None:
                    output+=tempo.right.char
                    tempo=root
                else:
                    tempo=tempo.right
    return output

n=int(input())
dict={}
for o in range(n):
    char,value=input().split()
    dict[char]=int(value)
Root=buildtree(dict)

while True:
    try:
        sample=input()
```

```
        if not sample:
            break
        if sample[0] in ("1","0"):
            print(release(Root,sample))
        else:
            em=""
            for m in sample:
                em+=find(Root)[m]
            print(em)
    except EOFError:#确保程序在用户结束输入后“正常退出”，而不是因为输入结束而“产生异常而中止”。try与except EOFError必须捆绑
        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:

实在是来不及理解了，理解完一定补上!

代码

```
#
```

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：试了一下字典，大数据输出不太对，感觉内存不太够？有时间再仔细想想

并查集，首先规定每个数的父节点就是自己。然后进行合并， 分别判断当前第一个数的父节点还是不是自己，如果不是，返回新的父节点。如果两个父节点都相同，那么不用做操作。否则，使二者的父节点相同。

代码

```
#
def getfather(x,father):
    if father[x]!=x:#说明此同学已经有同谋
        return getfather(father[x],father)#返回它的同谋
    return father[x]
def join(x,y,father):
    fx=getfather(x,father)#获得第一位同学的信仰
    fy=getfather(y,father)#获得第二位同学的信仰
    if fx==fy:#如果二者信仰相同，那么跳过
        return
    father[fx]=fy#第二位同学的信仰变为fx，二者加为同谋
csd=0
while True:
    try:
        csd+=1
        n,m=(int(x) for x in input().split())
        father = list(int(x) for x in range(n))  # 初始化自己的信仰
        if n==0:
            break
        for i in range(m):
            x,y=(int(x)for x in input().split())
            join(x-1,y-1,father)#由于编号问题要-1
        cnt=0
        for x in range(n):
            if father[x]==x:
                cnt+=1
        print(f"Case {csd}:",cnt)
    except EOFError:
        break
```

代码运行截图 ==（AC代码截图，至少包含有"Accepted"）==

CS101 / 题库

题目 排名 状态 提问

#44510807提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
def getfather(x,father):
    if father[x]!=x:#说明此同学已经有同谋
        return getfather(father[x],father)#返回它的同谋
    return father[x]
def join(x,y,father):
    fx=getfather(x,father)#获得第一位同学的信仰
    fy=getfather(y,father)#获得第二位同学的信仰
    if fx==fy:#如果二者信仰相同，那么跳过
        return
    father[fx]=fy#第二位同学的信仰变为fx，二者加为同谋
csd=0
while True:
    try:
        csd+=1
        n,m=(int(x) for x in input().split())
        father = list(int(x) for x in range(n))  # 初始化自己的信仰
        if n==0:
            break
        for i in range(m):
            x,y=(int(x)for x in input().split())
            join(x-1,y-1,father)#由于编号问题要-1
        cnt=0
        for x in range(n):
            if father[x]==x:
                cnt+=1
        print(f"Case {csd}:",cnt)
    except EOFError:
        break
```

基本信息

#: 44510807
题目: 02524
提交人: 23n2300013289
内存: 10784kB
时间: 1588ms
语言: Python3
提交时间: 2024-04-02 21:44:34

# 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。==

基本上内容就是树的高阶东西了，由于初学，对题解的理解花了很长的时间。

avl暂时没有时间看了，尽快补上

好难，虽然很想学好，但是自己能力有限，好笨，有点想退课了： （