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

Updated 2214 GMT+8 March 24, 2024

2024 spring, Complied by 赵语涵 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

操作系统：windows 11

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：开始没有弄懂二叉搜索树的规律。用分而治之的方法，把比节点小和大的分别放在节点两侧，并按后序表达式（先左子树再右子树最后节点）的形式表出

代码

```python
#赵语涵2300012254
n = int(input())
pre = list(map(int,input().split()))
def work(pre):
    if not pre:
        return []
    root = pre[0]
    left = [x for x in pre if x<root]
    right = [x for x in pre if x>root]
    return work(left)+work(right)+[root]
print(' '.join(map(str,work(pre))))
```

代码运行截图

#44473955提交状态

查看 提交 统计

状态: Accepted

源代码

```python
#赵语涵2300012254
n = int(input())
pre = list(map(int,input().split()))
def work(pre):
    if not pre:
        return []
    root = pre[0]
```

基本信息

#: 44473955
题目: 22275
提交人: 23n2300012254
内存: 3880kB
时间: 26ms
语言: Python3
提交时间: 2024-03-30 23:02:05

05455: 二叉搜索树的层次遍历

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

思路：与上题建造二叉搜索树的方式相比，为了方便后面的层次遍历，用树的数据类型储存。遍历时类似于BFS的思路，每一层的节点输出后，将其子节点存入下一层需要输出节点的列表里，知道子节点全为None为止

代码

```python
#赵语涵2300012254
pre = list(map(int,input().split()))
class Tree():
    def __init__(self,value):
        self.value = value
        self.left = None
        self.right = None
search_tree = []
def work(pre):
    if not pre:
        return None
    root = pre[0]
```

```
    t = Tree(root)
    t.left = work([x for x in pre if x<root])
    t.right = work([x for x in pre if x>root])
    search_tree.append(t)
    return t
work(pre)
result = []
nodes = [search_tree[-1]]
while nodes:
    new_nodes = []
    for i in nodes:
        result.append(i.value)
        if i.left:
            new_nodes.append(i.left)
        if i.right:
            new_nodes.append(i.right)
    nodes = new_nodes
print(' '.join(map(str,result)))
```

代码运行截图

#44474562提交状态 查看 提交 统计

状态: Accepted

源代码

```
#赵语涵2300012254
pre = list(map(int,input().split()))
class Tree():
    def __init__(self,value):
        self.value = value
        self.left = None
        self.right = None
```

基本信息

#: 44474562
题目: 05455
提交人: 23n2300012254
内存: 3632kB
时间: 25ms
语言: Python3
提交时间: 2024-03-31 00:27:26

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：按照自己的理解先写了，根据课件改进，相比起来对于上浮和下沉的过程没有用函数，而是放在add和delet里面循环。需要注意

代码

```
#赵语涵2300012254
class Heap():
    def __init__(self):
        self.size = 0
        self.data = [0]
```

```
    def add(self,x):
        self.data.append(x)
        self.size += 1
        current = self.size
        while (parent:=current//2) >0:
            if self.data[parent]>self.data[current]:
                self.data[parent],self.data[current] =
self.data[current],self.data[parent]
            current = parent

    def delet(self):
        print(self.data[1])
        self.data[1] = self.data[-1]
        self.data.pop()
        self.size -= 1
        i = 1
        while i*2 <= self.size:###
            if 2*i+1>self.size:
                min_i = 2*i
            else:
                if self.data[2*i]<self.data[2*i+1]:
                    min_i = 2*i
                else:
                    min_i = 2*i+1
            if self.data[i]>self.data[min_i]:
                self.data[i],self.data[min_i] =
self.data[min_i],self.data[i]
            i = min_i

heap = Heap()
for _ in range(int(input())):
    ope = input()
    if ope[0]=='1':
        heap.add(int(ope[2:]))
    else:
        heap.delet()
```

代码运行截图

#44509085提交状态

查看 提交 统计

状态: Accepted

源代码

```
#赵语涵2300012254
class Heap():
    def __init__(self):
        self.size = 0
        self.data = [0]

    def add(self,x):
```

基本信息

#: 44509085
题目: 04078
提交人: 23n2300012254
内存: 4696kB
时间: 564ms
语言: Python3
提交时间: 2024-04-02 19:51:26

22161: 哈夫曼编码树

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

思路：主要有建立哈夫曼编码树，编码，解码3种函数。

其中建立树的数据类型时注意：1）树有参数weight记录节点权重；2）为了使用heapq时通过比较树的权重得到树的排序，改写内置的__lt__\函数，使大小比较变为树的权重比较（heapq内置使用<进行排序）。

编码函数时弹出每个字母，从根节点开始遍历到节点下无子节点为止并记录节点的路径为结果。

解码时弹出每个数字直到按该数字代表路径没有子树为止即找到所编码的字母。

代码

```
#赵语涵2300012254
import heapq
class Tree():
    def __init__(self,x,y):
        self.weight = y
        self.name = x
        self.left = None
        self.right = None
    def __lt__(self,another):
        if self.weight < another.weight:
            return True
        else:
            return False
def new_name(x,y):
    if x[0]<y[0]:
        return x+y
    else:
        return y+x

def code(word):
    global new_trees
    result = ''
    while word:
        x = word.pop()
        start = root
        while start.left or start.right:
            if x in start.left.name:
                result += '0'
                start = start.left
            else:
                result += '1'
                start = start.right
    return result

def uncode(num,start):
    if not num:
        return start.name
```

```
    side = num.pop()
    result = ''
    if side == '0':
        if start.left:
            result += uncode(num,start.left)
        else:
            num.append(side)
            result = start.name + uncode(num,root)
    else:
        if start.right:
            result += uncode(num,start.right)
        else:
            num.append(side)
            result = start.name + uncode(num,root)
    return result


trees = []
new_trees = []
heapq.heapify(trees)
for _ in range((n:=int(input()))):
    data = list(input().split())
    heapq.heappush(trees, (t:=Tree(data[0],int(data[1]))))
    new_trees.append(t)
for _ in range(n-1):
    a,b = heapq.heappop(trees),heapq.heappop(trees)
    new_n = new_name(a.name, b.name)
    t = Tree(new_n,a.weight+b.weight)
    t.left,t.right = a,b
    heapq.heappush(trees, t)
    new_trees.append(t)
root = new_trees[-1]
while True:
    try:
        inp = list(input())[::-1]
        if inp[0] in ['0','1']:
            print(uncode(inp,root))
        else:
            print(code(inp))
    except EOFError:
        break
```

代码运行截图

#44501640提交状态

查看 提交 统计

状态: Accepted

源代码

```
#赵语涵2300012254
import heapq
class Tree():
    def __init__(self,x,y):
        self.weight = y
        self.name = x
        self.left = None
```

基本信息

#: 44501640
题目: 22161
提交人: 23n2300012254
内存: 3736kB
时间: 26ms
语言: Python3
提交时间: 2024-04-01 23:14:55

晴问9.5: 平衡二叉树的建立

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

思路：写的时候有多麻烦最后写完输出正确答案并整理注释代码时就有多愉快...这一部分的知识并没有提前非常了解，看教材加自己摸索收获很大

代码

```
#赵语涵2300012254
class Node():          #建立节点
    def __init__(self,value,left=None,right=None,parent=None,balance=0):
        self.value = value
        self.left = left
        self.right = right
        self.parent = parent
        self.balance = balance

    def isleft(self):
        return self.parent and self.parent.left==self

    def isright(self):
        return self.parent and self.parent.right==self

first = True
class Avltree():       #建立AVL树
    def __init__(self):
        self.root = None

    def put(self,x):     #put对每个数字建立Node并放入AVL树中
        global first
        if first:
            first = False  #第一个节点为根
            self.root = Node(x)
        else:
            self.childput(self.root,x)  #不是根节点，调用childput按照二叉搜索树规则放入新节点
    def childput(self,x,child):
        if child < x.value:
            if x.left:
                self.childput(x.left,child)
            else:
                x.left = Node(child,parent=x)
                self.balance(x.left)  #放入新节点后调用balance进行平衡
        elif child > x.value:
            if x.right:
                self.childput(x.right, child)
            else:
                x.right = Node(child,parent=x)
```

```
                self.balance(x.right)  #放入新节点后调用balance进行平衡

    def balance(self,node):
        if node.balance < -1 or node.balance > 1: #当前节点不符合要求，需要调用rebalance旋转
            self.rebalance(node)
            return
        if node.parent:  #当前节点符合平衡要求，向上遍历节点直到找到不符合要求或者到根节点为止
            if node.isleft():
                node.parent.balance += 1
            elif node.isright():
                node.parent.balance -= 1
            if node.parent.balance != 0:
                self.balance(node.parent)
    def rebalance(self,node):  #旋转要求：当需要左旋时先保证右子节点不左倾，反之亦然
        if node.balance > 1:
            if node.left.balance < 0:
                self.left_rotate(node.left)
            self.right_rotate(node)
        if node.balance < -1:
            if node.right.balance > 0:
                self.right_rotate(node.right)
            self.left_rotate(node)
    def left_rotate(self,rot):
        temp = rot.right
        if not rot.parent:  #是否改变根节点
            self.root = temp
        rot.right = temp.left
        if temp.left:
            temp.left.parent = rot
        if self.root == rot:
            self.root = temp
        else:
            if rot.isleft():
                rot.parent.left = temp
            elif rot.isright():
                rot.parent.right = temp
        temp.parent,rot.parent = rot.parent,temp
        temp.left = rot
        rot.balance = rot.balance+1-min(temp.balance,0) #根据balance=h(左)-h(右)推得
        temp.balance = temp.balance+1+max(rot.balance,0)
    def right_rotate(self,rot):
        temp = rot.left
        if not rot.parent:
            self.root = temp
        rot.left = temp.right
        if temp.right:
            temp.right.parent = rot
        if self.root == rot:
            self.root = temp
        else:
            if rot.isleft():
                rot.parent.left = temp
            elif rot.isright():
```

```
                rot.parent.right = temp
        temp.parent,rot.parent = rot.parent,temp
        temp.right = rot
        rot.balance = rot.balance-1-max(temp.balance,0)
        temp.balance = temp.balance-1+min(0,rot.balance)

result = []
def pre(x):  #前序输出结果
    global result
    result.append(x.value)
    if x.left:
        pre(x.left)
    if x.right:
        pre(x.right)

n = int(input())
t = Avltree()
for i in map(int,input().split()):
    t.put(i)
pre(t.root)
print(' '.join(map(str,result)))
```

代码运行截图

02524: 宗教信仰

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

思路：看课件和网上一些图解终于弄明白了disjoint set的用法。将节点分别引到根节点，给出2个应该同根的节点原来的根节点不同的话进行合并。最后计数根节点的数量

代码

```
#赵语涵2300012254
class Disjoint():
    def __init__(self,size):
        self.parent = [i for i in range(size)]
        self.rank = [0]*size
    def find(self,x):
        if self.parent[x] != x:
            self.parent[x] = self.find(self.parent[x])
        return self.parent[x]  #最终得到根节点的位置索引
    def union(self,x,y):
        root_x = self.find(x)
        root_y = self.find(y)
        if root_x != root_y:  #x、y的根节点索引不同，需要合并
            #优先选rank大的根节点进行合并，避免出现长链
            if self.rank[root_x] < self.rank[root_y]:
                self.parent[root_x] = root_y
            elif self.rank[root_x] > self.rank[root_y]:
                self.parent[root_y]  = root_x
            else:
                self.parent[root_x] = root_y  #将root_x位置的根节点指向root_y
                self.rank[root_y] += 1  #记录rank以达到最快查找的合并方式

c = 0
while True:
    c += 1
    n,m = map(int,input().split())
    if n == m == 0:
        break
    forest = Disjoint(n)
    for i in range(m):
        a,b = map(int,input().split())
        a,b = a-1,b-1
        forest.union(a,b)
    answer = {forest.find(x) for x in range(n)}
    print(f'Case {c}: {len(answer)}')
```

代码运行截图

#44512425提交状态 查看 提交 统计

状态: Accepted

源代码

```
#赵语涵2300012254
class Disjoint():
    def __init__(self,size):
        self.parent = [i for i in range(size)]
        self.rank = [0]*size
    def find(self,x):
        if self.parent[x] != x:
```

基本信息

#: 44512425
题目: 02524
提交人: 23n2300012254
内存: 6548kB
时间: 1480ms
语言: Python3
提交时间: 2024-04-03 02:27:51

2. 学习总结和收获

这周身体原因作业拖欠较多，余下部分稍后补齐

通过这次作业迅速学到了很多东西（感觉前期还是耽误了太多时间了，总之后面肯定需要好好消化一下）。对于构建类和类的方法通过本次作业的练习很大程度上加强了理解。