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

Updated 2214 GMT+8 March 24, 2024

2024 spring, Complied by 武昱达 23工院

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 2023.1.4 (Professional Edition)

1. 题目

22275: 二叉搜索树的遍历

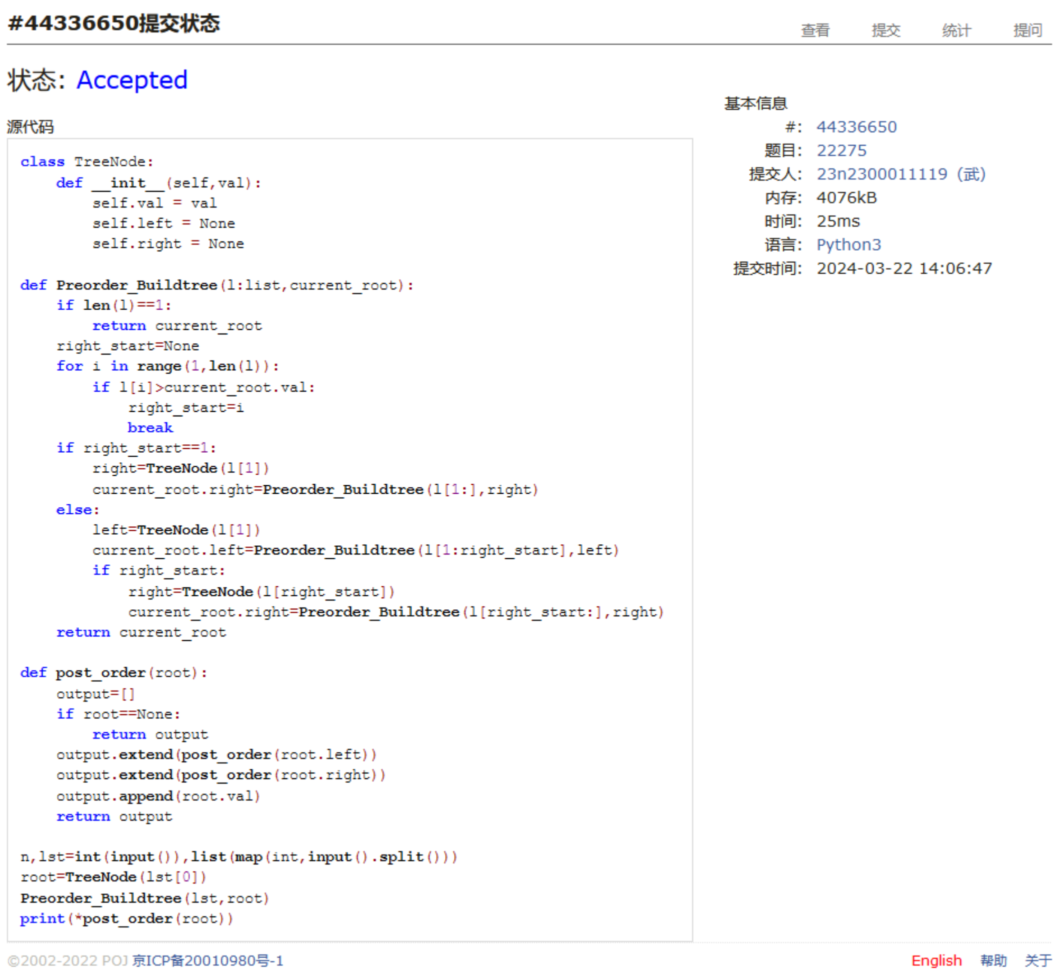
http://cs101.openjudge.cn/practice/22275/

思路：

代码

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

def Preorder_Buildtree(l:list,current_root):
    if len(l)==1:
        return current_root
    right_start=None
    for i in range(1,len(l)):
        if l[i]>current_root.val:
            right_start=i
            break
    if right_start==1:
        right=TreeNode(l[1])
        current_root.right=Preorder_Buildtree(l[1:],right)
    else:
        left=TreeNode(l[1])
        current_root.left=Preorder_Buildtree(l[1:right_start],left)
        if right_start:
            right=TreeNode(l[right_start])
            current_root.right=Preorder_Buildtree(l[right_start:],right)
    return current_root

def post_order(root):
    output=[]
    if root==None:
        return output
    output.extend(post_order(root.left))
    output.extend(post_order(root.right))
    output.append(root.val)
    return output

n,lst=int(input()),list(map(int,input().split()))
root=TreeNode(lst[0])
Preorder_Buildtree(lst,root)
print(*post_order(root))
```

代码运行截图 （至少包含有"Accepted"）



#44336650提交状态

查看 提交 统计 提问

状态: Accepted

源代码

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

def Preorder_Buildtree(l:list,current_root):
    if len(l)==1:
        return current_root
    right_start=None
    for i in range(1,len(l)):
        if l[i]>current_root.val:
            right_start=i
            break
    if right_start==1:
        right=TreeNode(l[1])
        current_root.right=Preorder_Buildtree(l[1:],right)
    else:
        left=TreeNode(l[1])
        current_root.left=Preorder_Buildtree(l[1:right_start],left)
        if right_start:
            right=TreeNode(l[right_start])
            current_root.right=Preorder_Buildtree(l[right_start:],right)
    return current_root

def post_order(root):
    output=[]
    if root==None:
        return output
    output.extend(post_order(root.left))
    output.extend(post_order(root.right))
    output.append(root.val)
    return output

n,lst=int(input()),list(map(int,input().split()))
root=TreeNode(lst[0])
Preorder_Buildtree(lst,root)
print(*post_order(root))
```

基本信息

#:	44336650
题目:	22275
提交人:	23n2300011119 (武)
内存:	4076kB
时间:	25ms
语言:	Python3
提交时间:	2024-03-22 14:06:47

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

05455: 二叉搜索树的层次遍历

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

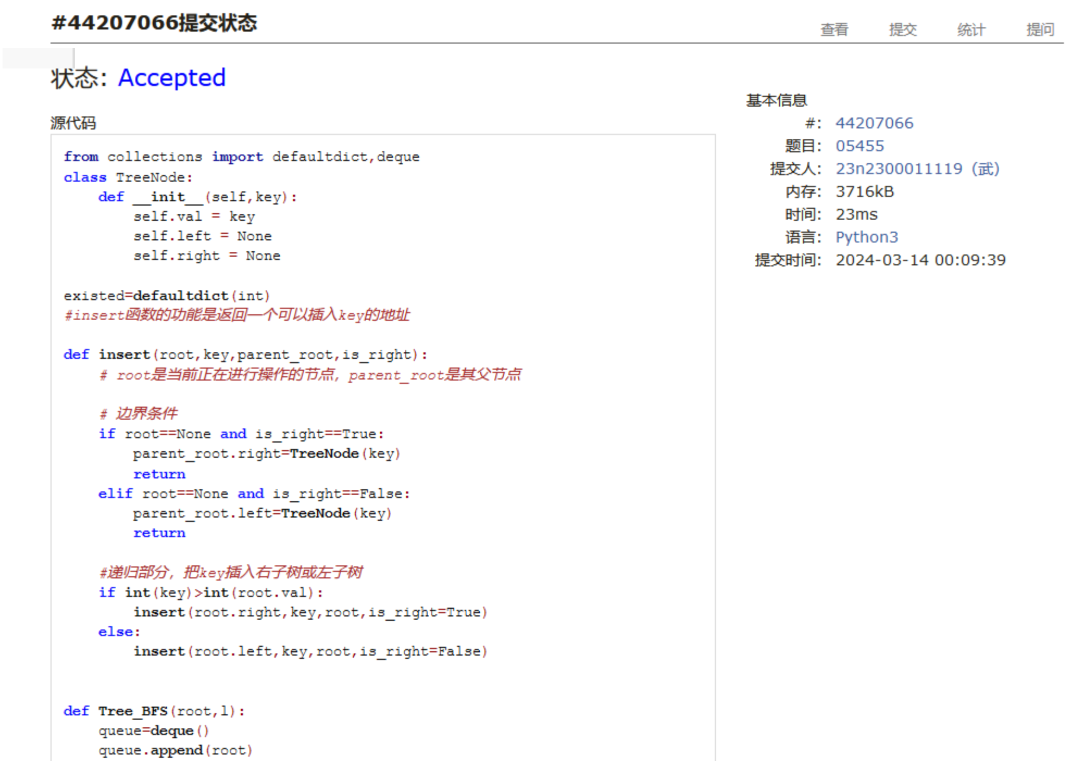
思路:

代码

```python
from collections import defaultdict,deque
class TreeNode:
    def __init__(self,key):
        self.val = key
        self.left = None
        self.right = None

```

```python
existed=defaultdict(int)
#insert函数的功能是返回一个可以插入key的地址

def insert(root,key,parent_root,is_right):
    # root是当前正在进行操作的节点，parent_root是其父节点

    # 边界条件
    if root==None and is_right==True:
        parent_root.right=TreeNode(key)
        return
    elif root==None and is_right==False:
        parent_root.left=TreeNode(key)
        return

    #递归部分，把key插入右子树或左子树
    if int(key)>int(root.val):
        insert(root.right,key,root,is_right=True)
    else:
        insert(root.left,key,root,is_right=False)


def Tree_BFS(root,l):
    queue=deque()
    queue.append(root)
    res=[root.val]
    while len(res)<l:
        a=queue.popleft()
        if a.left!=None:
            res.append(a.left.val)
            queue.append(a.left)
        if a.right!=None:
            res.append(a.right.val)
            queue.append(a.right)
    return res

raw=list(map(str,input().split()))
root=TreeNode(raw[0])
existed[raw[0]]=True
cnt=1
for i in raw[1:]:
    if not existed[i]:
        insert(root,i,None,True)
        cnt+=1
    existed[i]=1

print(" ".join(Tree_BFS(root,cnt)))
```

代码运行截图 （至少包含有"Accepted"）


#44207066提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```python
from collections import defaultdict,deque
class TreeNode:
    def __init__(self,key):
        self.val = key
        self.left = None
        self.right = None

existed=defaultdict(int)
#insert函数的功能是返回一个可以插入key的地址

def insert(root,key,parent_root,is_right):
    # root是当前正在进行操作的节点, parent_root是其父节点

    # 边界条件
    if root==None and is_right==True:
        parent_root.right=TreeNode(key)
        return
    elif root==None and is_right==False:
        parent_root.left=TreeNode(key)
        return

    #递归部分, 把key插入右子树或左子树
    if int(key)>int(root.val):
        insert(root.right,key,root,is_right=True)
    else:
        insert(root.left,key,root,is_right=False)


def Tree_BFS(root,l):
    queue=deque()
    queue.append(root)
```

基本信息

#:	44207066
题目:	05455
提交人:	23n2300011119 (武)
内存:	3716kB
时间:	23ms
语言:	Python3
提交时间:	2024-03-14 00:09:39

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：

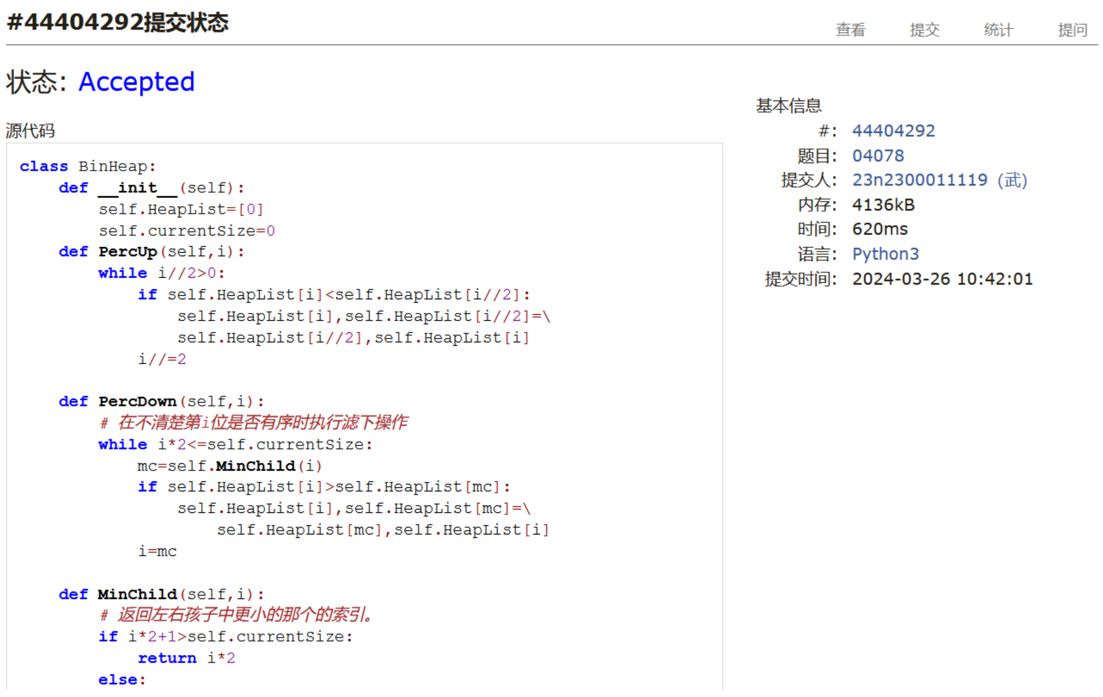
BinHeap类方法关键的两个函数就是PercDown和PercUp。他们实现了logn复杂度操作，依赖于完全二叉树的以下性质：

$$Node[i*2] == Node[i].left \qquad Node[i*2+1] == Node[i].right$$

代码

```python
class BinHeap:
    def __init__(self):
        self.HeapList=[0]
        self.currentSize=0
    def PercUp(self,i):
        while i//2>0:
            if self.HeapList[i]<self.HeapList[i//2]:
                self.HeapList[i],self.HeapList[i//2]=\
                self.HeapList[i//2],self.HeapList[i]
            i//=2

    def PercDown(self,i):
        # 在不清楚第i位是否有序时执行滤下操作
        while i*2<=self.currentSize:
            mc=self.MinChild(i)
            if self.HeapList[i]>self.HeapList[mc]:
                self.HeapList[i],self.HeapList[mc]=\
                    self.HeapList[mc],self.HeapList[i]
            i=mc

    def MinChild(self,i):
        # 返回左右孩子中更小的那个的索引。
        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 Insert(self,k):
        self.HeapList.append(k)
        self.currentSize+=1
        self.PercUp(self.currentSize)

    def DelMin(self):
        retval=self.HeapList[1]
        self.HeapList[1]=self.HeapList[self.currentSize]
        self.currentSize-=1
        self.HeapList.pop()
        self.PercDown(1)
        return retval

    def BuildHeap(self,alist):
        i=len(alist)//2
        self.currentSize=len(alist)
        self.HeapList.extend(alist[::])
        while i>0:
            self.PercDown(i)
            i-=1

lst=BinHeap()
for _ in range(int(input())):
    tmp=input()
    if tmp[0]=='1':
        a,b=map(int,tmp.split())
        lst.Insert(b)
    else:
        print(lst.DelMin())
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）



#44404292提交状态

状态: Accepted

源代码

```python
class BinHeap:
    def __init__(self):
        self.HeapList=[0]
        self.currentSize=0
    def PercUp(self,i):
        while i//2>0:
            if self.HeapList[i]<self.HeapList[i//2]:
                self.HeapList[i],self.HeapList[i//2]=\
                self.HeapList[i//2],self.HeapList[i]
            i//=2

    def PercDown(self,i):
        # 在不清楚第i位是否有序时执行滤下操作
        while i*2<=self.currentSize:
            mc=self.MinChild(i)
            if self.HeapList[i]>self.HeapList[mc]:
                self.HeapList[i],self.HeapList[mc]=\
                    self.HeapList[mc],self.HeapList[i]
            i=mc

    def MinChild(self,i):
        # 返回左右孩子中更小的那个的索引。
        if i*2+1>self.currentSize:
            return i*2
        else:
```

基本信息

#:	44404292
题目:	04078
提交人:	23n2300011119 (武)
内存:	4136kB
时间:	620ms
语言:	Python3
提交时间:	2024-03-26 10:42:01

22161: 哈夫曼编码树

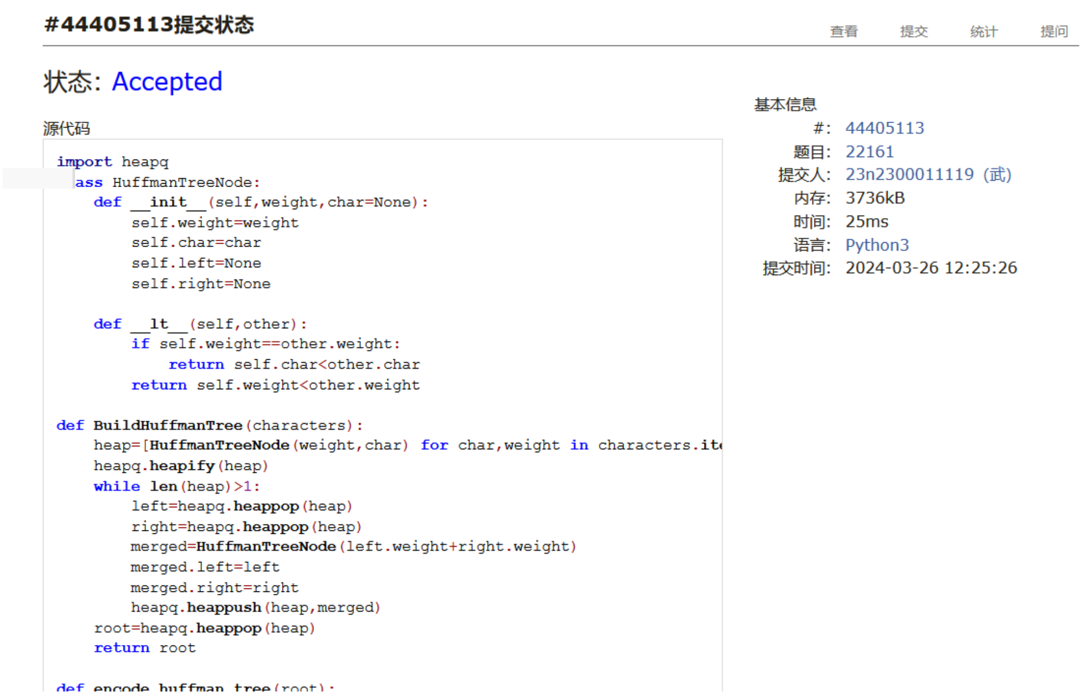
http://cs101.openjudge.cn/practice/22161/

思路:

代码

```python
import heapq
class HuffmanTreeNode:
    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 BuildHuffmanTree(characters):
    heap=[HuffmanTreeNode(weight,char) for char,weight in
characters.items()]
    heapq.heapify(heap)
    while len(heap)>1:
        left=heapq.heappop(heap)
        right=heapq.heappop(heap)
        merged=HuffmanTreeNode(left.weight+right.weight)
        merged.left=left
        merged.right=right
        heapq.heappush(heap,merged)
    root=heapq.heappop(heap)
    return root

def encode_huffman_tree(root):
    codes={}
    def traverse(node,code):
        if node.char:
            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.char:
            decoded+=node.char
            node=root
    return decoded

characters,strings,res={},[],[]
for _ in range(int(input())):
    char,weight=input().split()
    characters[char]=int(weight)
huffman_tree_root=BuildHuffmanTree(characters)
codes=encode_huffman_tree(huffman_tree_root)

while True:
    try:strings.append(input())
    except EOFError:break
for string in strings:
    if string.isnumeric():
        res.append(huffman_decoding(huffman_tree_root,string))
    else:
        res.append(huffman_encoding(codes,string))
for i in res:
    print(i)
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）



#44405113提交状态

状态: Accepted

源代码

```
import heapq
 ass HuffmanTreeNode:
    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 BuildHuffmanTree(characters):
    heap=[HuffmanTreeNode(weight,char) for char,weight in characters.ite
    heapq.heapify(heap)
    while len(heap)>1:
        left=heapq.heappop(heap)
        right=heapq.heappop(heap)
        merged=HuffmanTreeNode(left.weight+right.weight)
        merged.left=left
        merged.right=right
        heapq.heappush(heap,merged)
    root=heapq.heappop(heap)
    return root

def encode_huffman_tree(root):
```

基本信息

#:	44405113
题目:	22161
提交人:	23n2300011119 (武)
内存:	3736kB
时间:	25ms
语言:	Python3
提交时间:	2024-03-26 12:25:26

晴问9.5: 平衡二叉树的建立

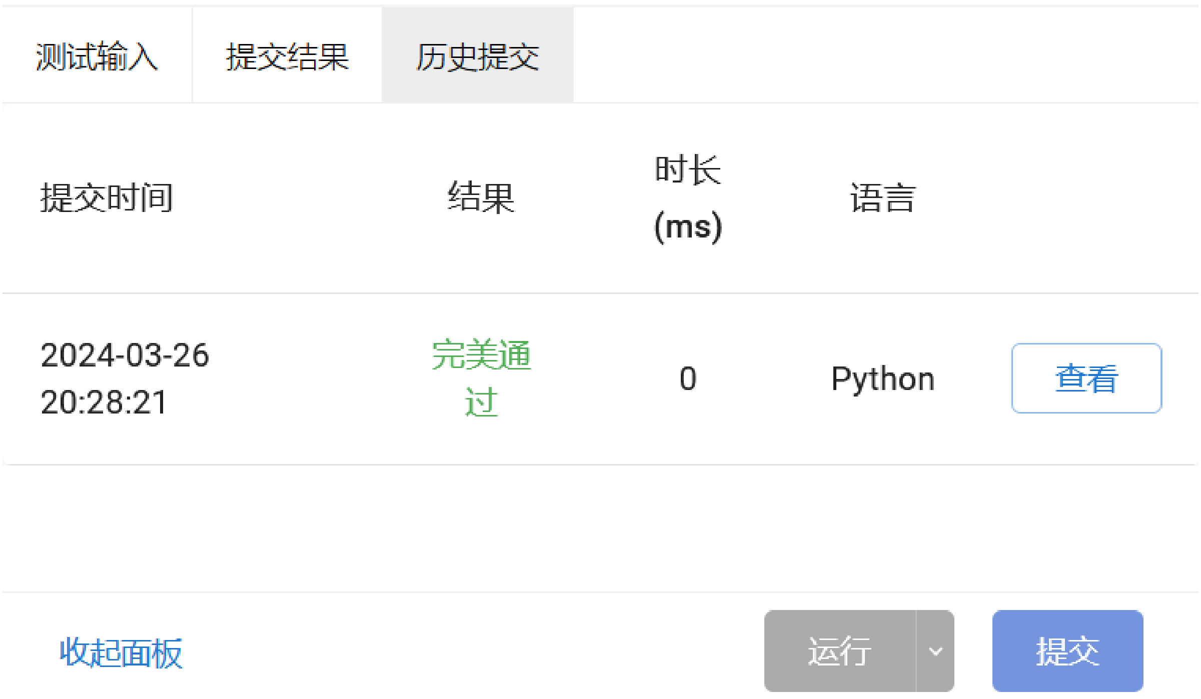
https://sunnywhy.com/sfbj/9/5/359

思路:

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"）



提交时间	结果	时长(ms)	语言	
2024-03-26 20:28:21	完美通过	0	Python	查看

02524: 宗教信仰

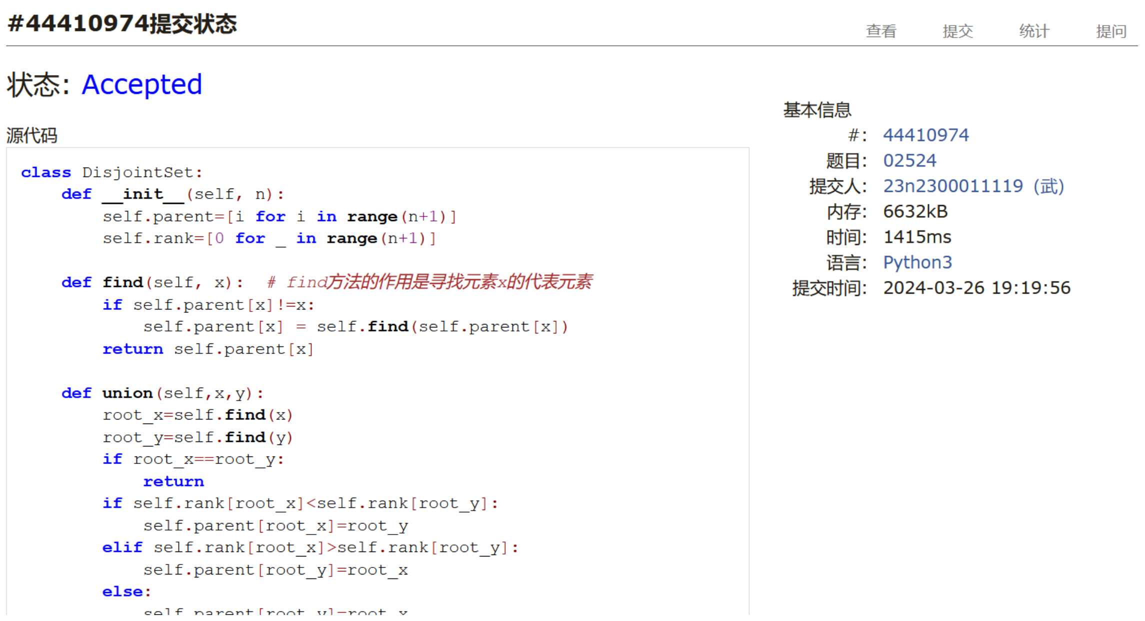
http://cs101.openjudge.cn/practice/02524/

思路：

代码

```python
class DisjointSet:
    # 实际上是用index作为每个元素的储存位置。
    def __init__(self, n):
        self.parent=[i for i in range(n+1)]
        self.rank=[0 for _ in range(n+1)]

    def find(self, x):  # find方法的作用是寻找元素x的代表元素
        if self.parent[x]!=x:
            # 注意，在递归地寻找父元素时，每一步操作并不浪费。
            # 我们递归地把跨越两层的路径压缩成跨越1层路径，这样能有效减少后续递归层数。
            self.parent[x] = self.find(self.parent[x])
        return self.parent[x]

    def union(self,x,y):
        # 找到各自元素的代表元素
        root_x=self.find(x)
        root_y=self.find(y)
        # 如果代表元素相同，说明属于一个集合，两元素无需合并
        if root_x==root_y:
            return
        # 如果一个的秩更大，那么把另一个元素挂到他的下面
        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
        # 如果秩一样大，那么把一个合并到另一个后，根的秩要+1
        else:
            self.parent[root_y]=root_x
            self.rank[root_x]+=1

num=1
while True:
    root_set=set()
    n,m=map(int,input().split())
    if n==0 and m==0:break
    DJS=DisjointSet(n)

    for _ in range(m):
        i,j=map(int,input().split())
        DJS.union(i,j)

    for i in range(1,n+1):
        root_set.add(DJS.find(i))

    print("Case {}: {}".format(num,len(root_set)))
    num+=1
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）



#44410974提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```python
class DisjointSet:
    def __init__(self, n):
        self.parent=[i for i in range(n+1)]
        self.rank=[0 for _ in range(n+1)]

    def find(self, x):  # find方法的作用是寻找元素x的代表元素
        if self.parent[x]!=x:
            self.parent[x] = self.find(self.parent[x])
        return self.parent[x]

    def union(self,x,y):
        root_x=self.find(x)
        root_y=self.find(y)
        if root_x==root_y:
            return
        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_y]=root_x
```

基本信息

#:	44410974
题目:	02524
提交人:	23n2300011119 (武)
内存:	6632kB
时间:	1415ms
语言:	Python3
提交时间:	2024-03-26 19:19:56

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。

本次作业有一部分是参考题解写出来的，惊奇地发现题解代码的优越性。

比如：

1. __lt__比较方法，大幅优化代码
2. 以_function命名的类私有方法，有效避免混乱
3. 在题目复杂时写树可以对节点和树分别进行定义，即可以有class AVLTree（同时进行class AVLTreeNode）这种操作，封装清楚，简洁易懂。

最后：上周末有点忙，每日选做落下了三四道题，正在努力追赶。