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

Updated 2214 GMT+8 March 24, 2024

2024 spring, Complied by 田济维 物理学院

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

（python pycharm）

操作系统：macOS Ventura 13.4.1 (c)

Python编程环境：Spyder IDE 5.2.2, PyCharm 2023.1.4 (Professional Edition)

C/C++编程环境：Mac terminal vi (version 9.0.1424), g++/gcc (Apple clang version 14.0.3, clang-1403.0.22.14.1)

1. 题目

22275: 二叉搜索树的遍历

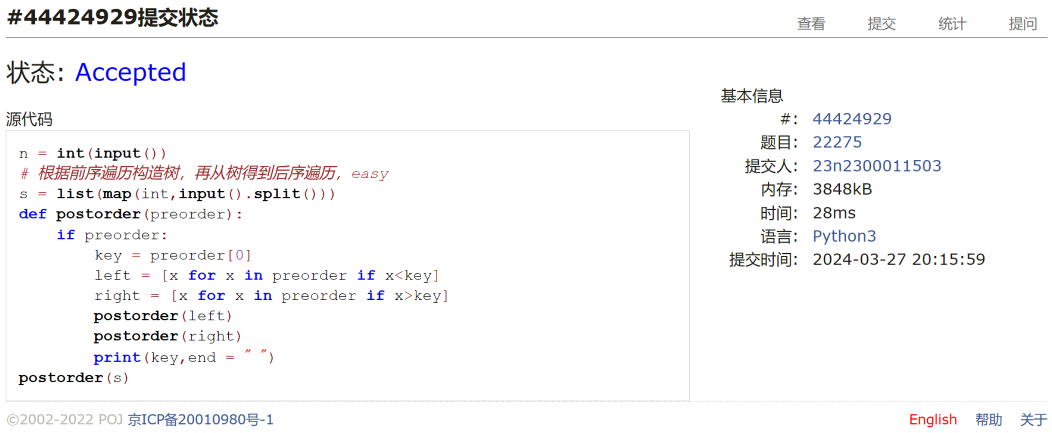
http://cs101.openjudge.cn/practice/22275/

思路：

代码

```python
#
n = int(input())
# 根据前序遍历构造树，再从树得到后序遍历，easy
s = list(map(int,input().split()))
def postorder(preorder):
    if preorder:
        key = preorder[0]
        left = [x for x in preorder if x<key]
        right = [x for x in preorder if x>key]
        postorder(left)
        postorder(right)
        print(key,end = " ")
postorder(s)
```

代码运行截图 （至少包含有"Accepted"）



#44424929提交状态

状态: Accepted

源代码

```python
n = int(input())
# 根据前序遍历构造树，再从树得到后序遍历，easy
s = list(map(int,input().split()))
def postorder(preorder):
    if preorder:
        key = preorder[0]
        left = [x for x in preorder if x<key]
        right = [x for x in preorder if x>key]
        postorder(left)
        postorder(right)
        print(key,end = " ")
postorder(s)
```

基本信息

#:	44424929
题目:	22275
提交人:	23n2300011503
内存:	3848kB
时间:	28ms
语言:	Python3
提交时间:	2024-03-27 20:15:59

©2002-2022 POJ 京ICP备20010980号-1

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 deque

class Node:
    def __init__(self,value):
        self.value = value
        self.left = None
        self.right = None


def insert(node, value):
    if node is None:
        return Node(value)
    if value < node.value:
        node.left = insert(node.left, value)
    elif value > node.value:
        node.right = insert(node.right, value)
    return node

def level_order_traversal(root):
    queue = [root]
    traversal = []
    while queue:
        node = queue.pop(0)
        traversal.append(node.value)
        if node.left:
            queue.append(node.left)
        if node.right:
            queue.append(node.right)
    return traversal
T= list(map(int,input().split()))
final =list(dict.fromkeys(T))

t = None
for x in final:
    t=insert(t,x)

print(" ".join(list(map(str,level_order_traversal(t)))))
```

代码运行截图 （至少包含有"Accepted"）

状态：Accepted

源代码

```
from collections import deque

class Node:
    def __init__(self,value):
        self.value = value
        self.left = None
        self.right = None


def insert(node, value):
    if node is None:
        return Node(value)
    if value < node.value:
        node.left = insert(node.left, value)
    elif value > node.value:
        node.right = insert(node.right, value)
    return node

def level_order_traversal(root):
    queue = [root]
    traversal = []
    while queue:
        node = queue.pop(0)
        traversal.append(node.value)
        if node.left:
            queue.append(node.left)
        if node.right:
            queue.append(node.right)
    return traversal
T= list(map(int,input().split()))
final =list(dict.fromkeys(T))

t = None
for x in final:
    t=insert(t,x)

print(" ".join(list(map(str,level_order_traversal(t)))))
```

04078: 实现堆结构

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

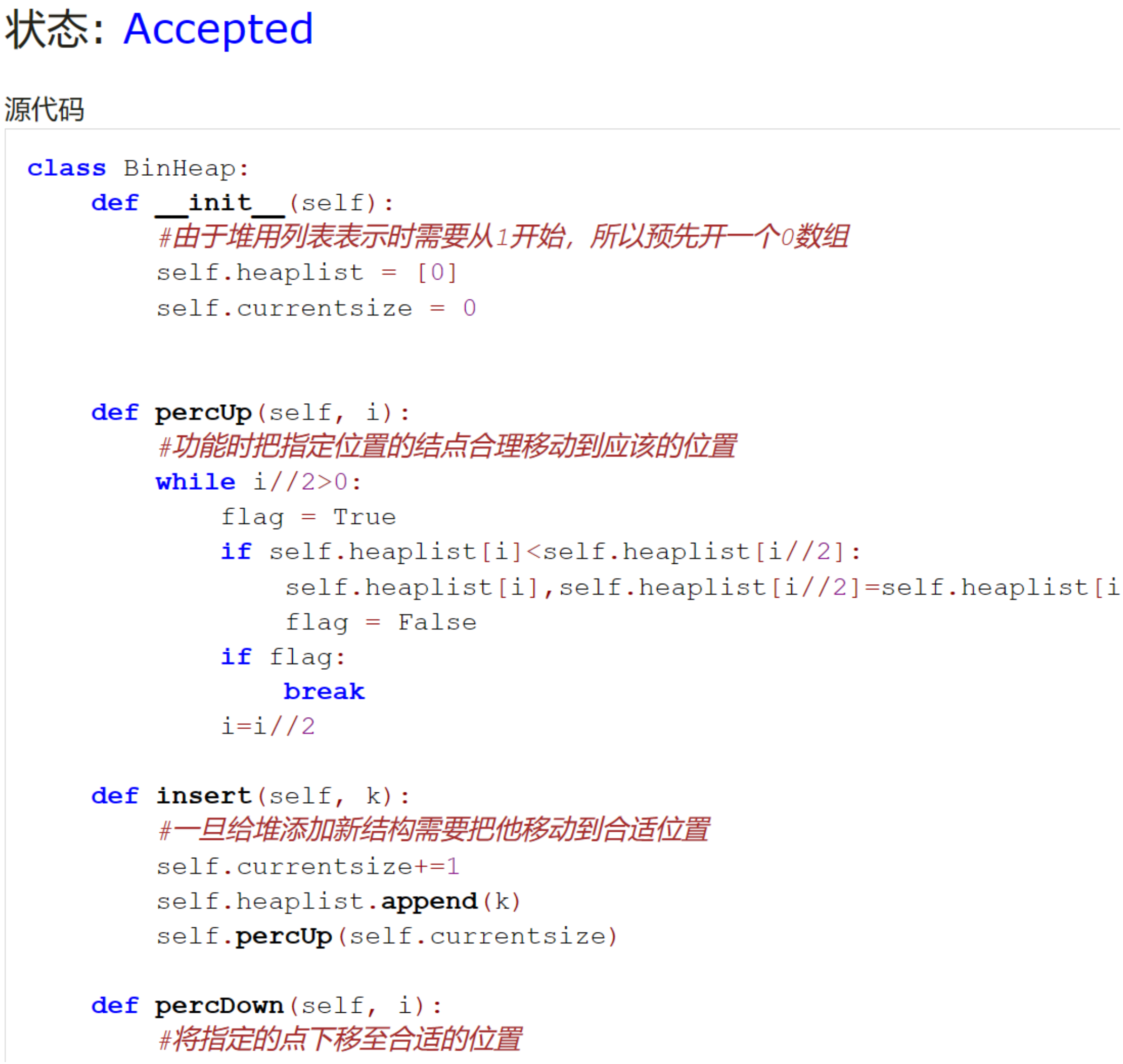
练习自己写个BinHeap。当然机考时候，如果遇到这样题目，直接import heapq。手搓栈、队列、堆、AVL等，考试前需要搓个遍。

思路：

代码

```python
#
class BinHeap:
    def __init__(self):
        #由于堆用列表表示时需要从1开始，所以预先开一个0数组
        self.heaplist = [0]
        self.currentsize = 0


    def percUp(self, i):
        #功能时把指定位置的结点合理移动到应该的位置
        while i//2>0:
            flag = True
            if self.heaplist[i]<self.heaplist[i//2]:
                self.heaplist[i],self.heaplist[i//2]=self.heaplist[i//2],self.heaplist[i]
                flag = False
            if flag:
                break
            i=i//2

    def insert(self, k):
        #一旦给堆添加新结构需要把他移动到合适位置
        self.currentsize+=1
        self.heaplist.append(k)
        self.percUp(self.currentsize)

    def percDown(self, i):
        #将指定的点下移至合适的位置
        while i*2<=self.currentsize:
            flag = True
            mc = self.minChild(i)
            if self.heaplist[i]>self.heaplist[mc]:
                self.heaplist[i],self.heaplist[mc]=self.heaplist[mc],self.heaplist[i]
                flag = False
            if flag:
                break
            i = mc

    def minChild(self, i):
        # 找到指定结点的最小的那个子结点
        if 2*i+1>self.currentsize:
            return 2*i
        else:
            return 2*i if self.heaplist[2*i]<self.heaplist[2*i+1] else 2*i+1

    def delMin(self):
        #取出栈顶元素
        s = self.heaplist[1]
        self.heaplist[1]=self.heaplist[self.currentsize]
        self.currentsize-=1
        self.heaplist.pop()
        self.percDown(1)
        return s
        #小心此时堆内只剩一个元素的情况
    def buildHeap(self, alist):
        #给定列表，构建堆对象
        i = len(alist)//2
        self.currentsize = len(alist)
        self.heaplist = [0]+alist
        while i>0:
            self.percDown(i)
            i-=1
n = int(input().strip())
heap = BinHeap()
for i in range(n):
    s=input().strip()
    if s[0]=="1":
        heap.insert(int(s.split()[1]))
    else:
        print(heap.delMin())

```

代码运行截图 （AC代码截图，至少包含有"Accepted"）



状态: Accepted

源代码

```python
class BinHeap:
    def __init__(self):
        #由于堆用列表表示时需要从1开始，所以预先开一个0数组
        self.heaplist = [0]
        self.currentsize = 0


    def percUp(self, i):
        #功能时把指定位置的结点合理移动到应该的位置
        while i//2>0:
            flag = True
            if self.heaplist[i]<self.heaplist[i//2]:
                self.heaplist[i],self.heaplist[i//2]=self.heaplist[i
                flag = False
            if flag:
                break
            i=i//2

    def insert(self, k):
        #一旦给堆添加新结构需要把他移动到合适位置
        self.currentsize+=1
        self.heaplist.append(k)
        self.percUp(self.currentsize)

    def percDown(self, i):
        #将指定的点下移至合适的位置
```

22161: 哈夫曼编码树

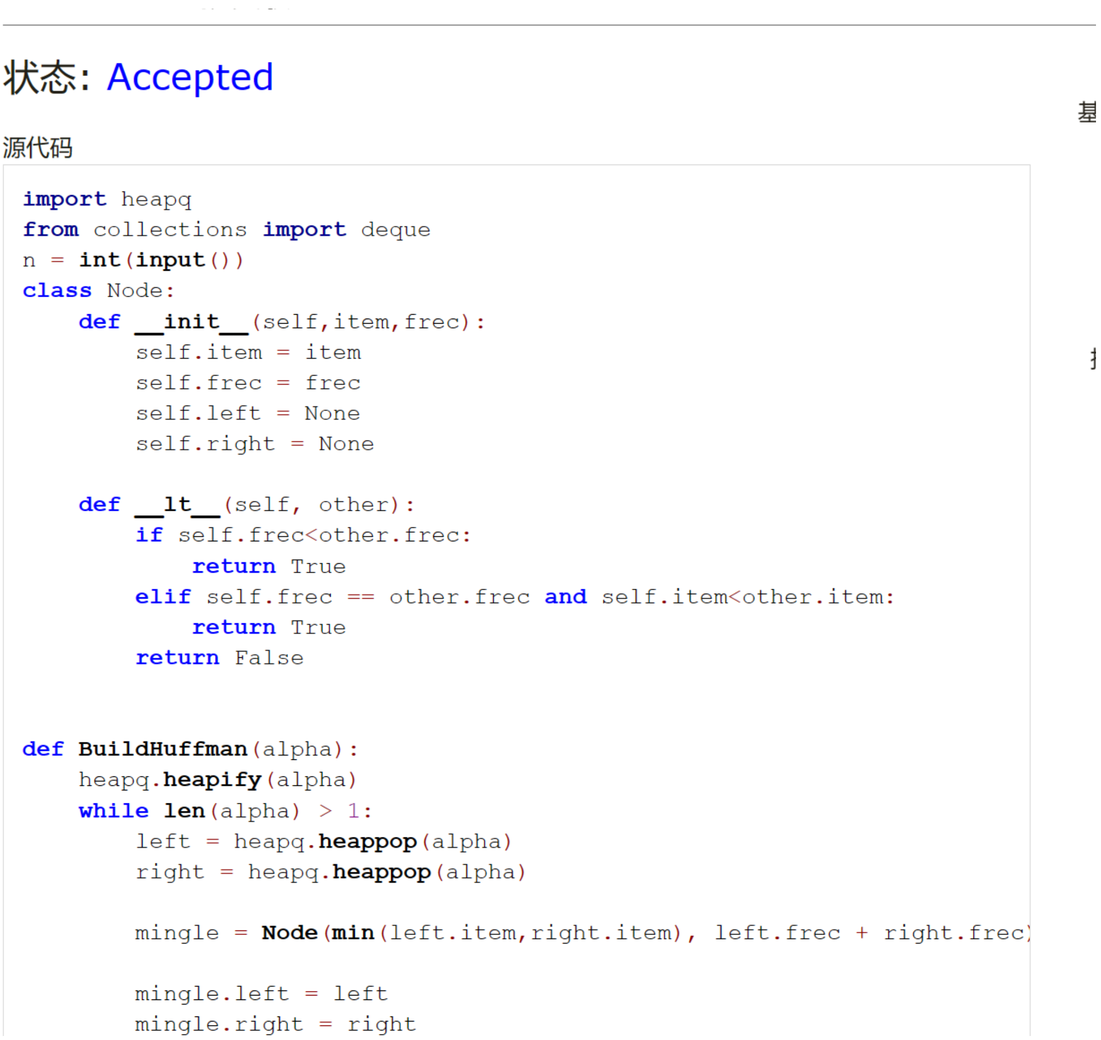
http://cs101.openjudge.cn/practice/22161/

思路：

代码

```python
#
import heapq
from collections import deque
n = int(input())
class Node:
    def __init__(self,item,frec):
        self.item = item
        self.frec = frec
        self.left = None
        self.right = None

    def __lt__(self, other):
        if self.frec<other.frec:
            return True
        elif self.frec == other.frec and self.item<other.item:
            return True
        return False


def BuildHuffman(alpha):
    heapq.heapify(alpha)
    while len(alpha) > 1:
        left = heapq.heappop(alpha)
        right = heapq.heappop(alpha)

        mingle = Node(min(left.item,right.item), left.frec + right.frec)

        mingle.left = left
        mingle.right = right
        alpha.append(mingle)
    return alpha[0]

def bfs(tree,item):
    que = deque([[tree,""]])
    while que:
        s = que.popleft()
        L=s[0].left
        R = s[0].right

        if L and R:
            if L.left == None:
                if L.item == item:
                    return s[1]+"0"
            else:
                que.append([L,s[1]+"0"])
            if R.left == None:
                if R.item == item:
                    return s[1]+"1"
            else:
                que.append([R,s[1]+"1"])

alpha = []
for i in range(n):
    s,frec = input().split()

    alpha.append(Node(s,int(frec)))

Huff = BuildHuffman(alpha)
while True:
    try:
        expr = input()
    except EOFError:
        break
    else:
        if expr[0].isdigit():
            result = []
            current = Huff
            for x in expr:
                if x == "0":
                    current = current.left
                else:
                    current = current.right
                if current.left == None:
                    result.append(current.item)
                    current = Huff
            print("".join(result))
        else:
            result = []
            for x in expr:
                result.append(bfs(Huff,x))
            print("".join(result))
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）



状态: Accepted

源代码

```
import heapq
from collections import deque
n = int(input())
class Node:
    def __init__(self,item,frec):
        self.item = item
        self.frec = frec
        self.left = None
        self.right = None

    def __lt__(self, other):
        if self.frec<other.frec:
            return True
        elif self.frec == other.frec and self.item<other.item:
            return True
        return False


def BuildHuffman(alpha):
    heapq.heapify(alpha)
    while len(alpha) > 1:
        left = heapq.heappop(alpha)
        right = heapq.heappop(alpha)

        mingle = Node(min(left.item,right.item), left.frec + right.frec)

        mingle.left = left
        mingle.right = right
```

晴问9.5: 平衡二叉树的建立

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

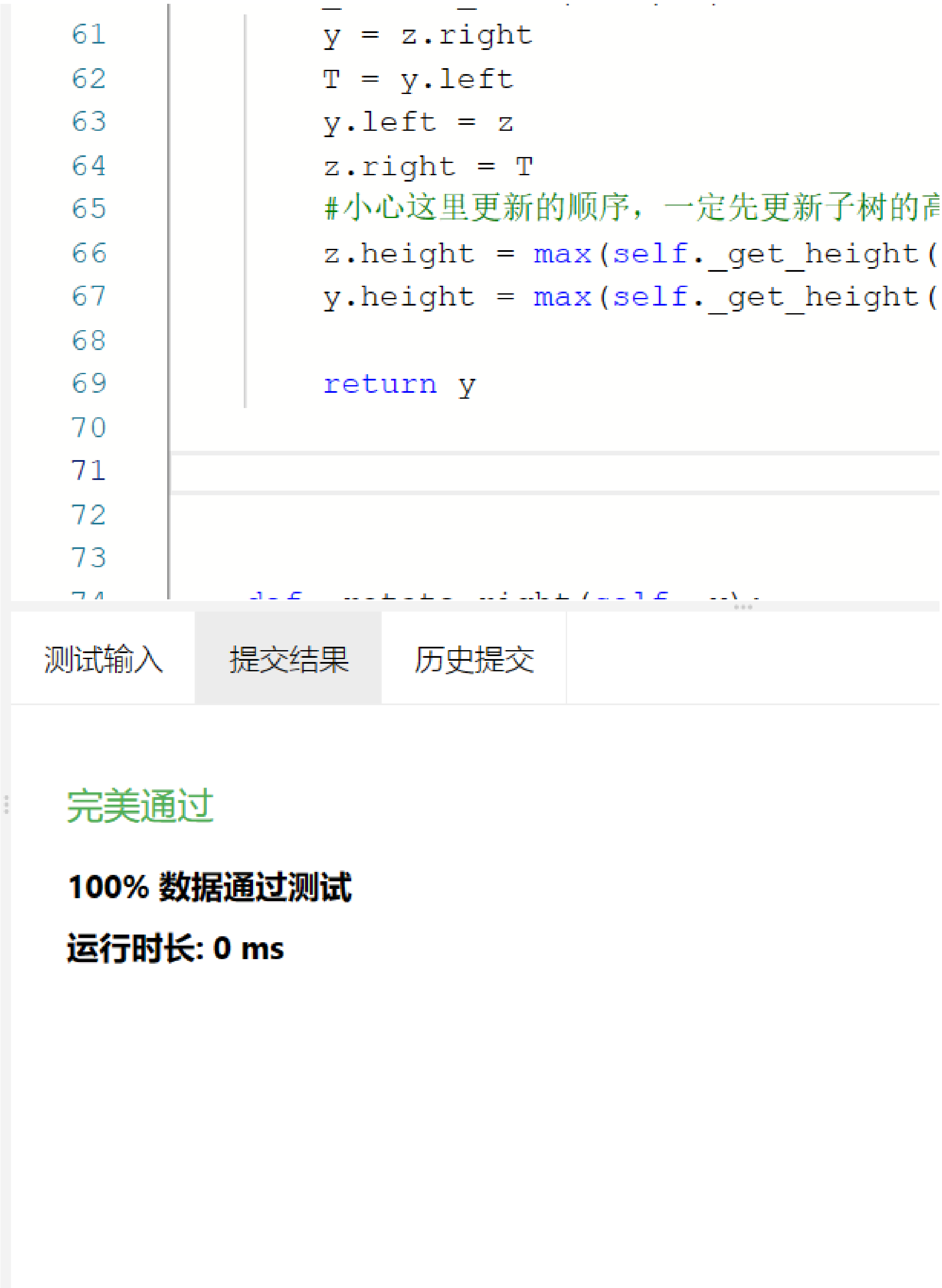
思路:

代码

```
#

class Node:
    def __init__(self,value):
        self.value = value
        self.left = None
        self.right = None
        self.height = 1
        #所有结点刚加入树的时候，一定是高度为1的

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

    def insert(self, value):
        if not self.root:
```

```python
            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:
            # 去平衡的插入左子树
            node.left = self._insert(value,node.left)
        else:
            node.right = self._insert(value,node.right)
        # 目前为止，node的字节点已经平衡好了，开始考察node本身的平衡
        balance = self._get_balance(node)
        node.height = 1+max(self._get_height(node.left),self._get_height(node.right))
        if balance>1:
            #初步判定是L
            if value<node.left.value:
                #LL
                return self._rotate_right(node)
            else:
                node.left = self._rotate_left(node.left)
                return  self._rotate_right(node)
        elif balance<-1:
            if value>=node.right.value:
                return self._rotate_left(node)
            else:
                node.right=self._rotate_right(node.right)
                return  self._rotate_left(node)
        return node



    def _get_height(self, node):
        if node:
            return node.height
        else:
            return 0

    def _get_balance(self, node):
        if node:
            return self._get_height(node.left)-self._get_height(node.right)


    def _rotate_left(self, z):
        y = z.right
        T = y.left
        y.left = z
        z.right = T
        #小心这里更新的顺序，一定先更新子树的高度
        z.height = max(self._get_height(z.left), self._get_height(z.right)) + 1
        y.height = max(self._get_height(y.left),self._get_height(y.right))+1

        return y




    def _rotate_right(self, y):
        z = y.left
        T = z.right
        z.right = y
        y.left = T
        y.height = max(self._get_height(y.left), self._get_height(y.right)) + 1
        z.height = max(self._get_height(z.left), self._get_height(z.right)) + 1
        return z



    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"）



```
        y = z.right
        T = y.left
        y.left = z
        z.right = T
        #小心这里更新的顺序，一定先更新子树的高
        z.height = max(self._get_height(
        y.height = max(self._get_height(

        return y



    def _rotate_right(self, y):
```

完美通过

100% 数据通过测试

运行时长: 0 ms

02524: 宗教信仰

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

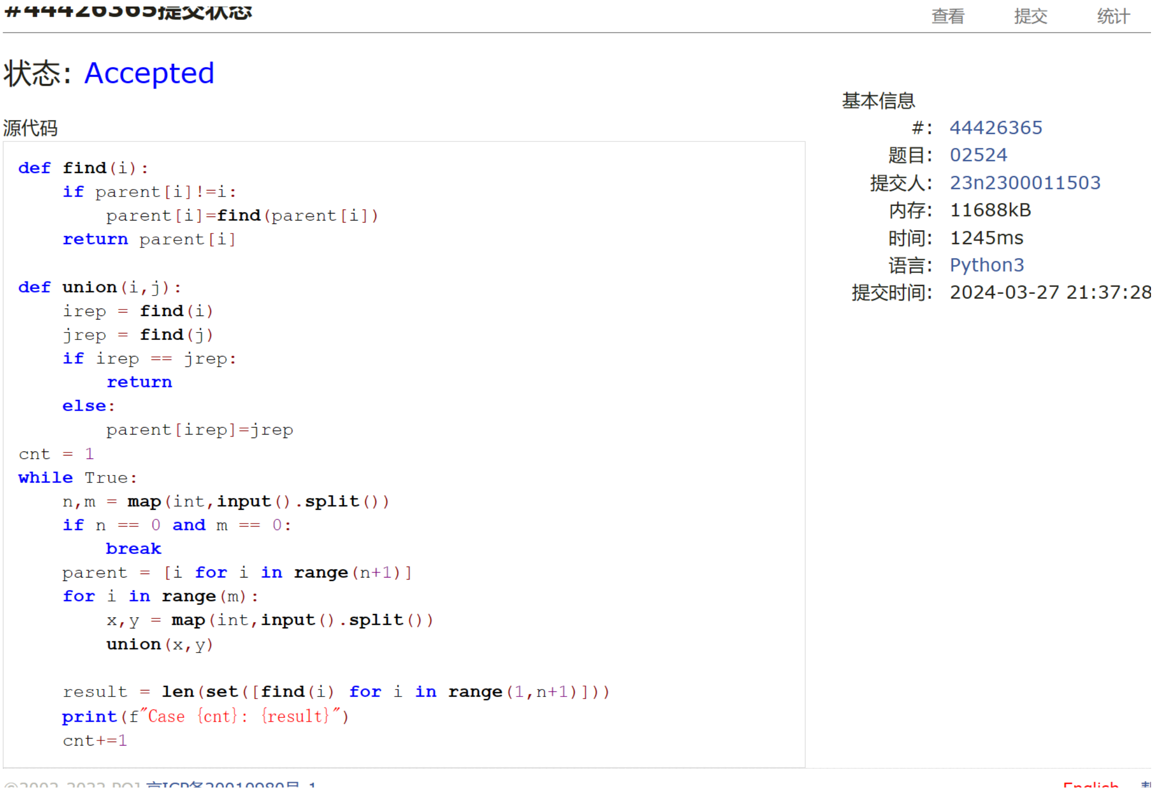
思路：

代码

```
#
```

```

from collections import deque
"""
class Node:
    def __init__(self,value):
        self.value = value
        self.left = None
        self.right = None


def insert(node, value):
    if node is None:
        return Node(value)
    if value < node.value:
        node.left = insert(node.left, value)
    elif value > node.value:
        node.right = insert(node.right, value)
    return node

def level_order_traversal(root):
    queue = [root]
    traversal = []
    while queue:
        node = queue.pop(0)
        traversal.append(node.value)
        if node.left:
            queue.append(node.left)
        if node.right:
            queue.append(node.right)
    return traversal
T= list(map(int,input().split()))
final =list(dict.fromkeys(T))

t = None
for x in final:
    t=insert(t,x)

print(" ".join(list(map(str,level_order_traversal(t)))))
"""
def find(i):
    if parent[i]!=i:
        parent[i]=find(parent[i])
    return parent[i]

def union(i,j):
    irep = find(i)
    jrep = find(j)
    if irep == jrep:
        return
    else:
        parent[irep]=jrep
cnt = 1
while True:
    n,m = map(int,input().split())
    if n == 0 and m == 0:
        break
    parent = [i for i in range(n+1)]
    for i in range(m):
        x,y = map(int,input().split())
        union(x,y)

    result = len(set([find(i) for i in range(1,n+1)]))
    print(f"Case {cnt}: {result}")
    cnt+=1
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）



状态: Accepted

源代码

```
def find(i):
    if parent[i]!=i:
        parent[i]=find(parent[i])
    return parent[i]

def union(i,j):
    irep = find(i)
    jrep = find(j)
    if irep == jrep:
        return
    else:
        parent[irep]=jrep
cnt = 1
while True:
    n,m = map(int,input().split())
    if n == 0 and m == 0:
        break
    parent = [i for i in range(n+1)]
    for i in range(m):
        x,y = map(int,input().split())
        union(x,y)

    result = len(set([find(i) for i in range(1,n+1)]))
    print(f"Case {cnt}: {result}")
    cnt+=1
```

基本信息

#:	44426365
题目:	02524
提交人:	23n2300011503
内存:	11688kB
时间:	1245ms
语言:	Python3
提交时间:	2024-03-27 21:37:28

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。

是目前为止收获最大的一节课和一次作业，学习了以前只听过名字或者只知道怎末用的数据类型，现在知道了他们的实现。并且很多计算概论时的题目变的清晰了，比如剪绳子就是哈夫曼编码，食物链就是并查集