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

Updated 2214 GMT+8 March 24, 2024

2024 spring, Complied by 同学的姓名、院系

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：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：

手搓一个二叉搜索树，把数据填进去，考虑到前序表达式有非常好的顺序性，直接让key等于列表第一个元素，剩下的数据放到left里直到列表空或数比root大，剩下的就是right，如果遇到空列表就直接嵌入对应的数值即可；构建树之后递归遍历即可

代码

```
class node:
    def __init__(self,s):
        self.key=s
        self.left=None
        self.right=None
```

```
    def post_traversal(self):
        traversal=[]
        if self.left:
            traversal+=self.left.post_traversal()
        if self.right:
            traversal+=self.right.post_traversal()
        traversal.append(self.key)
        return traversal

def build_tree(alst):
    if alst:
        root=alst.pop(0)
        left=[]
        while alst and alst[0]<root:
            left.append(alst.pop(0))
        right=alst
        current_node=node(root)
        current_node.left=build_tree(left)
        current_node.right=build_tree(right)
        return current_node
    else:
        return None

n=int(input())
pre_lst=[int(x) for x in input().split()]
tree_node=build_tree(pre_lst)
anslst=tree_node.post_traversal()
ansstr=[]
for x in anslst:
    ansstr.append(str(x))
print(' '.join(ansstr))
```

代码运行截图 （至少包含有"Accepted"）

#44499499提交状态

查看 提交 统计 提问

状态: Accepted

源代码

```
class node:
    def __init__(self,s):
        self.key=s
        self.left=None
        self.right=None

    def post_traversal(self):
        traversal=[]
        if self.left:
            traversal+=self.left.post_traversal()
        if self.right:
            traversal+=self.right.post_traversal()
        traversal.append(self.key)
        return traversal

def build_tree(alst):
    if alst:
        root=alst.pop(0)
        left=[]
        while alst and alst[0]<root:
            left.append(alst.pop(0))
        right=alst
        current_node=node(root)
        current_node.left=build_tree(left)
        current_node.right=build_tree(right)
        return current_node
    else:
        return None

n=int(input())
pre_lst=[int(x) for x in input().split()]
tree_node=build_tree(pre_lst)
anslst=tree_node.post_traversal()
ansstr=[]
for x in anslst:
    ansstr.append(str(x))
print(' '.join(ansstr))
```

基本信息

#: 44499499
题目: 22275
提交人: zxk
内存: 4088kB
时间: 29ms
语言: Python3
提交时间: 2024-04-01 20:18:39

English 帮助 关于

05455: 二叉搜索树的层次遍历

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

思路:

嵌入的函数用递归，对于每个输入的数据，根据现有的value比较总可以确定在哪一侧，直到这个位置为None时，就作为节点填入；层次遍历运用了队列，将一个节点拆成多个节点，放入队列，根据先进先出，队列始终保持着层次遍历的顺序，直到队列为空

代码

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

def insert(current_node,ist_value):
    if not current_node:
        return node(ist_value)
    if ist_value<current_node.value:
        current_node.left=insert(current_node.left,ist_value)
```

```
    if ist_value>current_node.value:
        current_node.right=insert(current_node.right,ist_value)
    return current_node

def level_order_traversal(rootnode):
    queue=[rootnode]
    anslst=[]
    while queue:
        current_node=queue.pop(0)
        if current_node:
            queue.append(current_node.left)
            queue.append(current_node.right)
            anslst.append(str(current_node.value))
    return ' '.join(anslst)

iptlst=[int(x) for x in input().split()]
tree=node(iptlst[0])
iptlst.pop(0)
for x in iptlst:
    insert(tree,x)
print(level_order_traversal(tree))
```

代码运行截图 （至少包含有"Accepted"）

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：

构建堆，应用堆

代码

```
class BinHeap:
    def __init__(self):
        self.heapList = [0]
        self.currentSize = 0
    # 堆的属性：一个列表和堆的大小

    def percUp(self, i):
        while i // 2 > 0:
            if self.heapList[i] < self.heapList[i // 2]:
               tmp = self.heapList[i // 2]
               self.heapList[i // 2] = self.heapList[i]
               self.heapList[i] = tmp
            i = i // 2
```

```
    # 实现数据上行

    def insert(self, k):
        self.heapList.append(k)
        self.currentSize += 1
        self.percUp(self.currentSize)
    # 新加元素

    def percDown(self,i):
        while (i*2)<=self.currentSize:
            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
    #实现数据下行

    def minChild(self,i):
        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
        #返回左右更小的数值

    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
    #删除最小元素

    def buildHeap(self,alist):
        i=len(alist)//2
        self.currentSize=len(alist)
        self.heapList=[0]+alist
        while i>0:
            self.percDown(i)
            i=i-1
    #建堆

ntimes=int(input())
heap=BinHeap()
anslst=[]
for _ in range(ntimes):
    ipt=[int(x) for x in input().split()]
    if ipt[0]==1:
        heap.insert(ipt[1])
    elif ipt[0]==2:
        anslst.append(heap.delMin())
for x in anslst:
```

```
    print(x)
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

状态: Accepted

源代码

```
class BinHeap:
    def __init__(self):
        self.heapList = [0]
        self.currentSize = 0
    # 堆的属性：一个列表和堆的大小

    def percUp(self, i):
        while i // 2 > 0:
            if self.heapList[i] < self.heapList[i // 2]:
               tmp = self.heapList[i // 2]
               self.heapList[i // 2] = self.heapList[i]
               self.heapList[i] = tmp
            i = i // 2
    # 实现数据上行

    def insert(self, k):
        self.heapList.append(k)
        self.currentSize += 1
        self.percUp(self.currentSize)
    # 新加元素

    def percDown(self,i):
        while (i*2)<=self.currentSize:
            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
    #实现数据下行

    def minChild(self,i):
        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
        #返回左右更小的数值

    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
    #删除最小元素

    def buildHeap(self,alist):
        i=len(alist)//2
        self.currentSize=len(alist)
        self.heapList=[0]+alist
        while i>0:
            self.percDown(i)
            i=i-1
    #建堆

ntimes=int(input())
heap=BinHeap()
anslst=[]
for _ in range(ntimes):
    ipt=[int(x) for x in input().split()]
    if ipt[0]==1:
        heap.insert(ipt[1])
    elif ipt[0]==2:
        anslst.append(heap.delMin())
for x in anslst:
    print(x)
```

基本信息

#:	44493593
题目:	04078
提交人:	zxk
内存:	6752kB
时间:	525ms
语言:	Python3
提交时间:	2024-03-31 23:28:43

22161: 哈夫曼编码树

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

思路：

首先用heapq构建一个堆，然后通过循环合并建树，然后定义两个函数用于编码和解码

代码

```
import heapq
class Node:
    def __init__(self,s,n):
        self.char=s
        self.frequency=n
        self.left=None
        self.right=None
        self.flag='leaf'
    def __lt__(self, other):
        if self.frequency<other.frequency:
            return True
        elif self.frequency==other.frequency:
            return self.char<other.char
        else:
            return False

def BuildTree(heap):
    while len(heap)>1:
        left=heapq.heappop(heap)
        right=heapq.heappop(heap)
        if left<right:
            node_char=left.char
        else:
            node_char=right.char
        new_node=Node(node_char,left.frequency+right.frequency)
        new_node.left=left
        new_node.right=right
        new_node.flag='node'
        heapq.heappush(heap,new_node)
    return heap[0]

def decode(astr):
    alst=[]
    for x in astr:
        alst.append(x)
    anslst=[]
    current_node=HuffmanTree
    while alst:
        tmp=alst[0]
        alst.pop(0)
        if current_node.flag=='node':
            if tmp=='1':
                current_node=current_node.right
                if not alst:
```

```
                    anslst.append(current_node.char)
            else:
                current_node=current_node.left
                if not alst:
                    anslst.append(current_node.char)
        else:
            anslst.append(current_node.char)
            alst.insert(0,tmp)
            current_node=HuffmanTree
    return ''.join(anslst)

code_sheet={}
def build_code_sheet(node,current_code):
    if node.flag=='leaf':
        code_sheet[node.char]=current_code
    else:
        build_code_sheet(node.left,current_code+'0')
        build_code_sheet(node.right,current_code+'1')

def encode(astr):
    anslst=[]
    for x in astr:
        anslst.append(code_sheet[x])
    return ''.join(anslst)
n=int(input())
heap=[]
for _ in range(n):
    ipt=[x for x in input().split()]
    node=Node(ipt[0],int(ipt[1]))
    heap.append(node)
heapq.heapify(heap)
HuffmanTree=BuildTree(heap)
build_code_sheet(HuffmanTree,'')
ipt_lst=[]

while True:
    try:
        ipt_lst.append(str(input()))
    except EOFError:
       break
for ipt_str in ipt_lst:
    if ipt_str[0] in ['0','1']:
        print(decode(ipt_str))
    else:
        print(encode(ipt_str))
```

代码运行截图 （AC代码截图，至少包含有"Accepted"）

状态: Accepted

基本信息

#: 44498742
题目: 22161
提交人: zxk
内存: 7596kB
时间: 27ms
语言: Python3
提交时间: 2024-04-01 19:27:30

源代码

```
import heapq
class Node:
    def __init__(self,s,n):
        self.char=s
        self.frequency=n
        self.left=None
        self.right=None
        self.flag='leaf'
    def __lt__(self, other):
        if self.frequency<other.frequency:
            return True
        elif self.frequency==other.frequency:
            return self.char<other.char
        else:
            return False

def BuildTree(heap):
    while len(heap)>1:
        left=heapq.heappop(heap)
        right=heapq.heappop(heap)
        if left<right:
            node_char=left.char
        else:
            node_char=right.char
        new_node=Node(node_char,left.frequency+right.frequency)
        new_node.left=left
        new_node.right=right
        new_node.flag='node'
        heapq.heappush(heap,new_node)
    return heap[0]

def decode(astr):
    alst=[]
    for x in astr:
        alst.append(x)
    anslst=[]
    current_node=HuffmanTree
    while alst:
        tmp=alst[0]
        alst.pop(0)
        if current_node.flag=='node':
            if tmp=='1':
                current_node=current_node.right
                if not alst:
                    anslst.append(current_node.char)
            else:
                current_node=current_node.left
                if not alst:
                    anslst.append(current_node.char)
        else:
            anslst.append(current_node.char)
            alst.insert(0,tmp)
            current_node=HuffmanTree
    return ''.join(anslst)

code_sheet={}
def build_code_sheet(node,current_code):
    if node.flag=='leaf':
        code_sheet[node.char]=current_code
    else:
        build_code_sheet(node.left,current_code+'0')
        build_code_sheet(node.right,current_code+'1')

def encode(astr):
    anslst=[]
    for x in astr:
        anslst.append(code_sheet[x])
    return ''.join(anslst)
n=int(input())
heap=[]
for _ in range(n):
    ipt=[x for x in input().split()]
    node=Node(ipt[0],int(ipt[1]))
    heap.append(node)
heapq.heapify(heap)
HuffmanTree=BuildTree(heap)
build_code_sheet(HuffmanTree,'')
ipt_lst=[]
```

晴问9.5: 平衡二叉树的建立

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

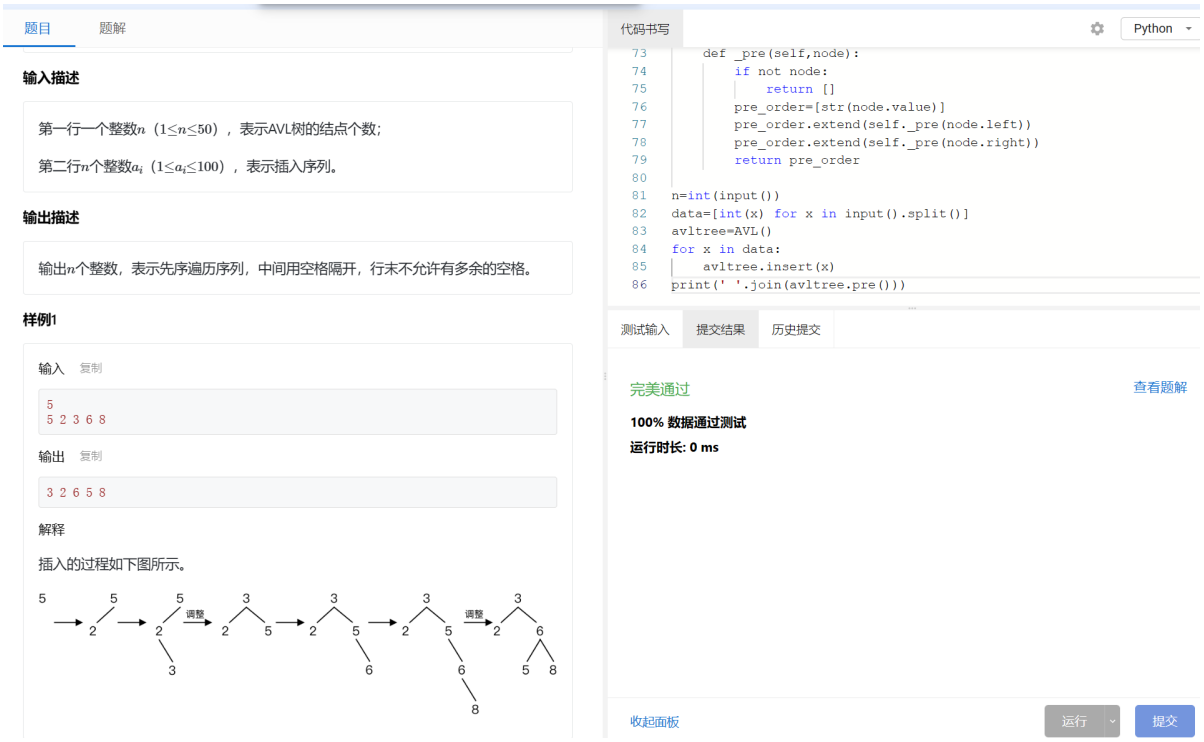
思路:

定义节点类、定义AVL类，AVL类中的方法有嵌入和旋转，最后返回一棵树，而AVL类中主要是定义了一些方法

代码

```
class Node:
    def __init__(self,val):
        self.value=val
        self.left=None
        self.right=None
        self.height=1

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

    def insert(self,val):
        if not self.root:
            self.root=Node(val)
        else:
            self.root=self._insert(val,self.root)

    def _get_height(self,node):
        if not node:
            return 0
        else:
            return node.height

    def _get_balance(self,node):
        if not node:
            return 0
        else:
            return self._get_height(node.left)-self._get_height(node.right)

    def _rotate_right(self,node):
        new_root=node.left
        node.left=new_root.right
        new_root.right=node

 node.height=1+max(self._get_height(node.left),self._get_height(node.right))

 new_root.height=1+max(self._get_height(new_root.left),self._get_height(new_root.
right))
        return new_root

    def _rotate_left(self,node):
        new_root=node.right
        node.right=new_root.left
```

```
        new_root.left=node
        node.height = 1 + max(self._get_height(node.left),
self._get_height(node.right))
        new_root.height = 1 + max(self._get_height(new_root.left),
self._get_height(new_root.right))
        return new_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:
                return self._rotate_right(node)
            else:
                node.left=self._rotate_left(node.left)
                return self._rotate_right(node)
        if 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 pre(self):
        return self._pre(self.root)
    def _pre(self,node):
        if not node:
            return []
        pre_order=[str(node.value)]
        pre_order.extend(self._pre(node.left))
        pre_order.extend(self._pre(node.right))
        return pre_order

n=int(input())
data=[int(x) for x in input().split()]
avltree=AVL()
for x in data:
    avltree.insert(x)
print(''.join(avltree.pre()))
```



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：

定义查找函数时候顺便把所有路径上的点全都变成根，简化查找时间；对于每个ij，分别查找各自的根，然后进行归并

代码

```
def find(x):
    if stu[x]!=x:
        stu[x]=find(stu[x])
    return stu[x]
case=0
while True:
    ipt=[int(x) for x in input().split()]
    if ipt==[0,0]:
        break
    case+=1
    n,m=ipt[0],ipt[1]
    stu=[i for i in range(n+1)]
    for _ in range(m):
        ijlst=[int(x) for x in input().split()]
        i,j=ijlst[0],ijlst[1]
        stu[find(i)]=find(j)
    s={find(x) for x in stu}
    print(f'Case {case}: {len(s)-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(x):
    if stu[x]!=x:
        stu[x]=find(stu[x])
    return stu[x]
case=0
while True:
    ipt=[int(x) for x in input().split()]
    if ipt==[0,0]:
        break
    case+=1
    n,m=ipt[0],ipt[1]
    stu=[i for i in range(n+1)]
    for _ in range(m):
        ijlst=[int(x) for x in input().split()]
        i,j=ijlst[0],ijlst[1]
        stu[find(i)]=find(j)
    s={find(x) for x in stu}
    print(f'Case {case}: {len(s)-1}')
```

基本信息

#:	44505131
题目:	02524
提交人:	zxk
内存:	11292kB
时间:	1243ms
语言:	Python3
提交时间:	2024-04-02

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。

·实现堆结构：看课件理解了堆结构，但是对heapq还没掌握（理解最快的一集），其实也不是很理解为什么这样做的时间复杂度会小很多，并且让我复述一遍如何实现也很难，还得靠包

·Huffman编码：确实不简单啊。。甚至还都没考虑编码时候的时间复杂度，靠遍历把所有的字符的编码都写了一边；用了很长时间，开始的时候对于heapq的各种函数都不熟悉，也分不清和自己编的heap有什么不一样；之后卡在了heapify的变量是数值列表，但是我想构建节点的堆，不明白怎么办，看了课件之后恍然明白了在定义节点类的时候可以定义比大小__lt__（网上查了一下其实不应该只定义小于，不过很多包比如heapq只用了小于，所以还好）；做好了heap列表之后卡在了建树，开始的时候其实并没明白哈夫曼树是一个什么结构，还在考虑左右节点是否都存在的问题，后来动手画了一下才明白其实和表达式有些类似，是一个root带两个node，root包含两个node的频率和最小字符，这样就可以用来和其他的node进行比较，类似于某个运算之后的结果会表示为一个运算符节点；解码的时候就循环一下，碰到leaf就写进答案，然后重置；编码的时候想了想对于查找某个char所在的路径以我目前的能力还做不到，所以干脆直接写一个编码表得了

·二叉搜索树：其实课件上没有建树，但是我懒得记住那些了hhh，课件上用的是递归，思路其实相当于把建树和后序遍历揉在了一起，不过既然我会建树，会后序遍历，干脆就直接默写！省的递归半天对那些判据捯饬不清

·二叉搜索树层次遍历：两个主要的函数都是参考了课件，自己手写的，对于二叉搜索树这个数据类型理解起来比较简单，关于建树过程中的数据有一个左右可以一空一不空的问题，需要先加判断是否非空；这道题最大的收获是变量不要和类、属性、方法、函数等用同一个名字....否则会导致无法判断到底是什么（例如我对于一个用'node'定义的类，就不能写node=node(value)，否则对于'node.XXXX'将无法判断到底是用的类中某个属性还是对变量node用某个方法）

·宗教信仰：完全不会查并集，开始的时候还觉得用一个列表表示出现与否挺快的，看了数据之后呆了。。竟然有这么大数据量的题我天，果然一般的方法都不行，遂看答案，几乎是照抄，稍微一点不对就会wa，感觉核心在于怎样把没用的数据丢掉，即在find函数里让路径上的每一个节点的根都变成一个根，这样路径中出现过的数据就可以不用再经过了，也就是实现了把树高度减小但是宽度增大，最终留下固定数目的根节点

·AVL：基本等于放弃了，抄答案都不知道该怎么抄，原理倒是明白，代码实现完全不会...看了各种渠道的树，感觉都太复杂了，而且有各种父类子类啊、一个两个下划线啊、试着做了一个AVL，但是倒来倒去四个类四个旋转方法搞得连数据类型长什么样都搞不清了（然后写前序遍历的时候自己就知道肯定不对了。。。然后看了群里一个答案，才顺着把代码写完；虽然没有参考不可能写出来，但是收获还是挺多的，比如如果在第二个类中用了第一个类定义的数据类型，后边想根据第一个类型处理，可以在类中定义一个函数，这样即使是用了第二个类的self，实际上也是只对第一个类进行操作，返回的第一个类的数据类型即可；对于所谓的平衡因子，其实直接用高度相减就可以，没必要重新定义一个平衡因子，算起来还麻烦；对于在二叉搜索树中嵌入元素的过程，叶节点和其他节点的类型不太一样，所以在各个方法中都要判断是否为叶，就解决了遍历等方法到叶节点的报错。

总体来说还是感觉很难，各种算法各种数据结构都快混了。。。