

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

---

Updated 2214 GMT+8 March 24, 2024

2024 spring, Compiled by 田济维 物理学院

## 说明:

- 1) 这次作业内容不简单, 耗时长直接参考题解。
- 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

```

1  #
2  n = int(input())
3  # 根据前序遍历构造树，再从树得到后序遍历，easy
4  s = list(map(int,input().split()))
5  def postorder(preorder):
6      if preorder:
7          key = preorder[0]
8          left = [x for x in preorder if x<key]
9          right = [x for x in preorder if x>key]
10         postorder(left)
11         postorder(right)
12         print(key,end = " ")
13     postorder(s)

```

代码运行截图 (至少包含有"Accepted")

#44424929提交状态

[查看](#) [提交](#) [统计](#) [提问](#)

状态: Accepted

源代码

```

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

[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

```

1  #
2  from collections import deque
3
4  class Node:
5      def __init__(self,value):
6          self.value = value
7          self.left = None
8          self.right = None
9
10
11  def insert(node, value):
12      if node is None:

```

```

13         return Node(value)
14     if value < node.value:
15         node.left = insert(node.left, value)
16     elif value > node.value:
17         node.right = insert(node.right, value)
18     return node
19
20 def level_order_traversal(root):
21     queue = [root]
22     traversal = []
23     while queue:
24         node = queue.pop(0)
25         traversal.append(node.value)
26         if node.left:
27             queue.append(node.left)
28         if node.right:
29             queue.append(node.right)
30     return traversal
31 T= list(map(int,input().split()))
32 final =list(dict.fromkeys(T))
33
34 t = None
35 for x in final:
36     t=insert(t,x)
37
38 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

```

1  #
2  class BinHeap:
3      def __init__(self):
4          #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
5          self.heaplist = [0]
6          self.currentsize = 0
7
8
9      def percup(self, i):
10         #功能是把指定位置的结点合理移动到应该的位置
11         while i//2>0:
12             flag = True
13             if self.heaplist[i]<self.heaplist[i//2]:
14
15 self.heaplist[i],self.heaplist[i//2]=self.heaplist[i//2],self.heaplist[i]
16                 flag = False
17                 if flag:
18                     break
19                 i=i//2
20
21 def insert(self, k):
22     #一旦给堆添加新结构需要把他移动到合适位置
23     self.currentsize+=1
24     self.heaplist.append(k)
25     self.percup(self.currentsize)
26
27 def percdwn(self, i):
28     #将指定的点下移至合适的位置
29     while i*2<=self.currentsize:
30         flag = True
31         mc = self.minChild(i)
32         if self.heaplist[i]>self.heaplist[mc]:
33
34 self.heaplist[i],self.heaplist[mc]=self.heaplist[mc],self.heaplist[i]
35                 flag = False
36                 if flag:
37                     break
38                 i = mc
39
40 def minChild(self, i):
41     # 找到指定结点的最小的那个子结点
42     if 2*i+1>self.currentsize:
43         return 2*i
44     else:
45         return 2*i if self.heaplist[2*i]<self.heaplist[2*i+1] else 2*i+1
46
47 def delMin(self):
48     #取出栈顶元素
49     s = self.heaplist[1]
50     self.heaplist[1]=self.heaplist[self.currentsize]
51     self.currentsize-=1
52     self.heaplist.pop()
53     self.percDown(1)
54     return s
55     #小心此时堆内只剩一个元素的情况

```

```

54     def buildHeap(self, alist):
55         #给定列表, 构建堆对象
56         i = len(alist)//2
57         self.currentsize = len(alist)
58         self.heaplist = [0]+alist
59         while i>0:
60             self.percDown(i)
61             i-=1
62 n = int(input().strip())
63 heap = BinHeap()
64 for i in range(n):
65     s=input().strip()
66     if s[0]=="1":
67         heap.insert(int(s.split()[1]))
68     else:
69         print(heap.delMin())
70

```

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):
        #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: 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

```



```

45         que.append([L,s[1]+"0"])
46         if R.left == None:
47             if R.item == item:
48                 return s[1]+"1"
49         else:
50             que.append([R,s[1]+"1"])
51
52     alpha = []
53     for i in range(n):
54         s,frec = input().split()
55
56         alpha.append(Node(s,int(frec)))
57
58     Huff = BuildHuffman(alpha)
59     while True:
60         try:
61             expr = input()
62         except EOFError:
63             break
64         else:
65             if expr[0].isdigit():
66                 result = []
67                 current = Huff
68                 for x in expr:
69                     if x == "0":
70                         current = current.left
71                     else:
72                         current = current.right
73                     if current.left == None:
74                         result.append(current.item)
75                         current = Huff
76                 print("".join(result))
77             else:
78                 result = []
79                 for x in expr:
80                     result.append(bfs(Huff,x))
81                 print("".join(result))

```

代码运行截图 (AC代码截图, 至少包含有"Accepted")



状态: Accepted

基

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

```
1  #
2
3  class Node:
4      def __init__(self,value):
5          self.value = value
6          self.left = None
7          self.right = None
8          self.height = 1
9          #所有结点刚加入树的时候，一定是高度为1的
10
11 class AVL:
12     def __init__(self):
13         self.root = None
14
15     def insert(self, value):
16         if not self.root:
```

```

17         self.root = Node(value)
18     else:
19         self.root = self._insert(value, self.root)
20
21     def _insert(self, value, node):
22         if not node:
23             return Node(value)
24         elif value < node.value:
25             # 去平衡的插入左子树
26             node.left = self._insert(value, node.left)
27         else:
28             node.right = self._insert(value, node.right)
29             # 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
30             balance = self._get_balance(node)
31             node.height =
1+max(self._get_height(node.left), self._get_height(node.right))
32             if balance > 1:
33                 #初步判定是L
34                 if value < node.left.value:
35                     #LL
36                     return self._rotate_right(node)
37                 else:
38                     node.left = self._rotate_left(node.left)
39                     return self._rotate_right(node)
40             elif balance < -1:
41                 if value >= node.right.value:
42                     return self._rotate_left(node)
43                 else:
44                     node.right = self._rotate_right(node.right)
45                     return self._rotate_left(node)
46             return node
47
48
49
50     def _get_height(self, node):
51         if node:
52             return node.height
53         else:
54             return 0
55
56     def _get_balance(self, node):
57         if node:
58             return self._get_height(node.left) - self._get_height(node.right)
59
60
61     def _rotate_left(self, z):
62         y = z.right
63         T = y.left
64         y.left = z
65         z.right = T
66         #小心这里更新的顺序, 一定先更新子树的高度
67         z.height = max(self._get_height(z.left), self._get_height(z.right))
+ 1
68         y.height =
max(self._get_height(y.left), self._get_height(y.right)) + 1

```

```

69
70         return y
71
72
73
74
75     def _rotate_right(self, y):
76         z = y.left
77         T = z.right
78         z.right = y
79         y.left = T
80         y.height = max(self._get_height(y.left), self._get_height(y.right))
+ 1
81         z.height = max(self._get_height(z.left), self._get_height(z.right))
+ 1
82         return z
83
84
85
86     def preorder(self):
87         return self._preorder(self.root)
88     def _preorder(self, node):
89         if not node:
90             return []
91         return
[ node.value ] + self._preorder(node.left) + self._preorder(node.right)
92
93 n = int(input().strip())
94 sequence = list(map(int, input().strip().split()))
95
96 avl = AVL()
97 for value in sequence:
98     avl.insert(value)
99
100 print(' '.join(map(str, avl.preorder())))

```

代码运行截图 (AC代码截图, 至少包含有"Accepted")

```

61         y = z.right
62         T = y.left
63         y.left = z
64         z.right = T
65         #小心这里更新的顺序，一定先更新子树的高
66         z.height = max(self._get_height(
67             y.height = max(self._get_height(
68
69         return y
70
71
72
73
74

```

测试输入

提交结果

历史提交

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

1 #

```

2
3 from collections import deque
4 """
5 class Node:
6     def __init__(self,value):
7         self.value = value
8         self.left = None
9         self.right = None
10
11
12 def insert(node, value):
13     if node is None:
14         return Node(value)
15     if value < node.value:
16         node.left = insert(node.left, value)
17     elif value > node.value:
18         node.right = insert(node.right, value)
19     return node
20
21 def level_order_traversal(root):
22     queue = [root]
23     traversal = []
24     while queue:
25         node = queue.pop(0)
26         traversal.append(node.value)
27         if node.left:
28             queue.append(node.left)
29         if node.right:
30             queue.append(node.right)
31     return traversal
32 T= list(map(int,input().split()))
33 final =list(dict.fromkeys(T))
34
35 t = None
36 for x in final:
37     t=insert(t,x)
38
39 print(" ".join(list(map(str,level_order_traversal(t)))))
40 """
41 def find(i):
42     if parent[i]!=i:
43         parent[i]=find(parent[i])
44     return parent[i]
45
46 def union(i,j):
47     irep = find(i)
48     jrep = find(j)
49     if irep == jrep:
50         return
51     else:
52         parent[irep]=jrep
53 cnt = 1
54 while True:
55     n,m = map(int,input().split())
56     if n == 0 and m == 0:

```

```

57         break
58     parent = [i for i in range(n+1)]
59     for i in range(m):
60         x,y = map(int,input().split())
61         union(x,y)
62
63     result = len(set([find(i) for i in range(1,n+1)]))
64     print(f"Case {cnt}: {result}")
65     cnt+=1

```

代码运行截图 (AC代码截图, 至少包含有"Accepted")

#44426365提交记录

[查看](#) [提交](#) [统计](#)

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