

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

Updated 2214 GMT+8 March 24, 2024

2024 spring, Compiled 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

```
1 class TreeNode:
2     def __init__(self, val):
3         self.val = val
4         self.left = None
5         self.right = None
6
7 def Preorder_Buildtree(l: list, current_root):
8     if len(l) == 1:
9         return current_root
10    right_start = None
11    for i in range(1, len(l)):
12        if l[i] > current_root.val:
13            right_start = i
14            break
15    if right_start == 1:
16        right = TreeNode(l[1])
17        current_root.right = Preorder_Buildtree(l[1:], right)
18    else:
19        left = TreeNode(l[1])
20        current_root.left = Preorder_Buildtree(l[1:right_start], left)
21        if right_start:
22            right = TreeNode(l[right_start])
23            current_root.right = Preorder_Buildtree(l[right_start:], right)
24    return current_root
25
26 def post_order(root):
27     output = []
28     if root == None:
```

```

29         return output
30     output.extend(post_order(root.left))
31     output.extend(post_order(root.right))
32     output.append(root.val)
33     return output
34
35 n, lst = int(input()), list(map(int, input().split()))
36 root = TreeNode(lst[0])
37 Preorder_Buildtree(lst, root)
38 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

```

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

```

1 from collections import defaultdict, deque
2 class TreeNode:
3     def __init__(self, key):
4         self.val = key
5         self.left = None
6         self.right = None
7

```

```

8 existed=defaultdict(int)
9 #insert函数的功能是返回一个可以插入key的地址
10
11 def insert(root,key,parent_root,is_right):
12     # root是当前正在进行操作的节点，parent_root是其父节点
13
14     # 边界条件
15     if root==None and is_right==True:
16         parent_root.right=TreeNode(key)
17         return
18     elif root==None and is_right==False:
19         parent_root.left=TreeNode(key)
20         return
21
22     #递归部分，把key插入右子树或左子树
23     if int(key)>int(root.val):
24         insert(root.right,key,root,is_right=True)
25     else:
26         insert(root.left,key,root,is_right=False)
27
28
29 def Tree_BFS(root,l):
30     queue=deque()
31     queue.append(root)
32     res=[root.val]
33     while len(res)<1:
34         a=queue.popleft()
35         if a.left!=None:
36             res.append(a.left.val)
37             queue.append(a.left)
38         if a.right!=None:
39             res.append(a.right.val)
40             queue.append(a.right)
41     return res
42
43 raw=list(map(str,input().split()))
44 root=TreeNode(raw[0])
45 existed[raw[0]]=True
46 cnt=1
47 for i in raw[1:]:
48     if not existed[i]:
49         insert(root,i,None,True)
50         cnt+=1
51     existed[i]=1
52
53 print(" ".join(Tree_BFS(root,cnt)))

```

代码运行截图 (至少包含有"Accepted")

状态: Accepted

源代码

```

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:

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

代码

```

1 class BinHeap:
2     def __init__(self):
3         self.HeapList=[0]
4         self.currentSize=0
5     def PercUp(self,i):
6         while i//2>0:
7             if self.HeapList[i]<self.HeapList[i//2]:
8                 self.HeapList[i],self.HeapList[i//2]=\
9                     self.HeapList[i//2],self.HeapList[i]
10            i//=2
11
12    def PercDown(self,i):
13        # 在不清楚第i位是否有序时执行滤下操作
14        while i*2<=self.currentSize:
15            mc=self.MinChild(i)
16            if self.HeapList[i]>self.HeapList[mc]:
17                self.HeapList[i],self.HeapList[mc]=\

```

```

18         self.HeapList[mc],self.HeapList[i]
19         i=mc
20
21     def MinChild(self,i):
22         # 返回左右孩子中更小的那个的索引。
23         if i*2+1>self.currentSize:
24             return i*2
25         else:
26             if self.HeapList[i*2]<self.HeapList[i*2+1]:
27                 return i*2
28             else:return i*2+1
29
30     def Insert(self,k):
31         self.HeapList.append(k)
32         self.currentSize+=1
33         self.PercUp(self.currentSize)
34
35     def DelMin(self):
36         retval=self.HeapList[1]
37         self.HeapList[1]=self.HeapList[self.currentSize]
38         self.currentSize-=1
39         self.HeapList.pop()
40         self.PercDown(1)
41         return retval
42
43     def BuildHeap(self,alist):
44         i=len(alist)//2
45         self.currentSize=len(alist)
46         self.HeapList.extend(alist[:])
47         while i>0:
48             self.PercDown(i)
49             i-=1
50
51 lst=BinHeap()
52 for _ in range(int(input())):
53     tmp=input()
54     if tmp[0]=='1':
55         a,b=map(int,tmp.split())
56         lst.Insert(b)
57     else:
58         print(lst.DelMin())

```

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
    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

```
1 import heapq
2 class HuffmanTreeNode:
3     def __init__(self,weight,char=None):
4         self.weight=weight
5         self.char=char
6         self.left=None
7         self.right=None
8
9     def __lt__(self,other):
10         if self.weight==other.weight:
11             return self.char<other.char
12         return self.weight<other.weight
13
14 def BuildHuffmanTree(characters):
15     heap=[HuffmanTreeNode(weight,char) for char,weight in
16 characters.items()]
17     heapq.heapify(heap)
18     while len(heap)>1:
19         left=heapq.heappop(heap)
20         right=heapq.heappop(heap)
21         merged=HuffmanTreeNode(left.weight+right.weight)
22         merged.left=left
23         merged.right=right
24         heapq.heappush(heap,merged)
25     root=heapq.heappop(heap)
26     return root
27 def encode_huffman_tree(root):
```

```

28     codes={}
29     def traverse(node,code):
30         if node.char:
31             codes[node.char]=code
32         else:
33             traverse(node.left,code+'0')
34             traverse(node.right,code+'1')
35     traverse(root,"")
36     return codes
37
38 def huffman_encoding(codes,string):
39     encoded=""
40     for char in string:
41         encoded+=codes[char]
42     return encoded
43
44 def huffman_decoding(root,encoded_string):
45     decoded=""
46     node=root
47     for bit in encoded_string:
48         if bit=='0':
49             node=node.left
50         else:
51             node=node.right
52         if node.char:
53             decoded+=node.char
54             node=root
55     return decoded
56
57 characters,strings,res={},[],[]
58 for _ in range(int(input())):
59     char,weight=input().split()
60     characters[char]=int(weight)
61 huffman_tree_root=BuildHuffmanTree(characters)
62 codes=encode_huffman_tree(huffman_tree_root)
63
64 while True:
65     try:strings.append(input())
66     except EOFError:break
67 for string in strings:
68     if string.isnumeric():
69         res.append(huffman_decoding(huffman_tree_root,string))
70     else:
71         res.append(huffman_encoding(codes,string))
72 for i in res:
73     print(i)

```

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
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):
```



```

29     balance = self._get_balance(node)
30
31     if balance > 1:
32         if value < node.left.value: # 树形是 LL
33             return self._rotate_right(node)
34         else: # 树形是 LR
35             node.left = self._rotate_left(node.left)
36             return self._rotate_right(node)
37
38     if balance < -1:
39         if value > node.right.value: # 树形是 RR
40             return self._rotate_left(node)
41         else: # 树形是 RL
42             node.right = self._rotate_right(node.right)
43             return self._rotate_left(node)
44
45     return node
46
47 def _get_height(self, node):
48     if not node:
49         return 0
50     return node.height
51
52 def _get_balance(self, node):
53     if not node:
54         return 0
55     return (self._get_height(node.left) -
56            self._get_height(node.right))
57
58 def _rotate_left(self, z):
59     y = z.right
60     T2 = y.left
61     y.left = z
62     z.right = T2
63     z.height = 1 + max(self._get_height(z.left),
64                        self._get_height(z.right))
65     y.height = 1 + max(self._get_height(y.left),
66                        self._get_height(y.right))
67     return y
68
69 def _rotate_right(self, y):
70     x = y.left
71     T2 = x.right
72     x.right = y
73     y.left = T2
74     y.height = 1 + max(self._get_height(y.left),
75                        self._get_height(y.right))
76     x.height = 1 + max(self._get_height(x.left),
77                        self._get_height(x.right))
78     return x
79
80 def preorder(self):
81     return self._preorder(self.root)
82
83 def _preorder(self, node):
84     if not node:

```

```
85         return []
86         return ([node.value] + self._preorder(node.left) +
87                 self._preorder(node.right))
88
89
90 n = int(input().strip())
91 sequence = list(map(int, input().strip().split()))
92 avl = AVL()
93 for value in sequence:
94     avl.insert(value)
95 print(' '.join(map(str, avl.preorder())))
```

代码运行截图 (AC代码截图, 至少包含有"Accepted")

测试输入	提交结果	历史提交			
提交时间	结果	时长 (ms)	语言		
2024-03-26 20:28:21	完美通过	0	Python	<div>查看</div>	

收起面板

运行

提交

02524: 宗教信仰

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

思路:

代码

```
1 class DisjointSet:
2     # 实际上是用index作为每个元素的储存位置。
3     def __init__(self, n):
4         self.parent=[i for i in range(n+1)]
5         self.rank=[0 for _ in range(n+1)]
6
7     def find(self, x): # find方法的作用是寻找元素x的代表元素
8         if self.parent[x]!=x:
9             # 注意, 在递归地寻找父元素时, 每一步操作并不浪费。
10            # 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。
11            self.parent[x] = self.find(self.parent[x])
12        return self.parent[x]
13
```

```

14     def union(self,x,y):
15         # 找到各自元素的代表元素
16         root_x=self.find(x)
17         root_y=self.find(y)
18         # 如果代表元素相同，说明属于一个集合，两元素无需合并
19         if root_x==root_y:
20             return
21         # 如果一个的秩更大，那么把另一个元素挂到他的下面
22         if self.rank[root_x]<self.rank[root_y]:
23             self.parent[root_x]=root_y
24         elif self.rank[root_x]>self.rank[root_y]:
25             self.parent[root_y]=root_x
26         # 如果秩一样大，那么把一个合并到另一个后，根的秩要+1
27         else:
28             self.parent[root_y]=root_x
29             self.rank[root_x]+=1
30
31 num=1
32 while True:
33     root_set=set()
34     n,m=map(int,input().split())
35     if n==0 and m==0:break
36     DJS=DisjointSet(n)
37
38     for _ in range(m):
39         i,j=map(int,input().split())
40         DJS.union(i,j)
41
42     for i in range(1,n+1):
43         root_set.add(DJS.find(i))
44
45     print("Case {}: {}".format(num,len(root_set)))
46     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

```

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_x]=root_y
            self.rank[root_x]+=1

```

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。