

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

Updated GMT+8 March 30, 2024

2024 spring, Compiled by 钟俊宇 物理学院

编程环境

Windows 11 家庭中文版, PyCharm Community Edition 2023.3.3

1. 题目

22275: 二叉搜索树的遍历

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

思路:

根据二叉搜索树的性质建树, 然后进行前序遍历

代码

```

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

def buildtree(s):
    if len(s) == 0:
        return None
    root = Treenode(s[0])
    left = []
    right = []
    for i in range(1, len(s)):
        if s[i] < s[0]:
            left.append(s[i])
        if s[i] > s[0]:
            right.append(s[i])
    root.left = buildtree(left)
    root.right = buildtree(right)
    return root

def post(node: Treenode):
    if not node:
        return []
    out = []
    out.extend(post(node.left))
    out.extend(post(node.right))
    out.append(str(node.value))
    return out

n = int(input())
str0 = list(map(int, input().split()))
print(' '.join(post(buildtree(str0))))

```

代码运行截图

状态: **Accepted**

源代码

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

def buildtree(s):
    if len(s) == 0:
        return None
    root = Treenode(s[0])
    left = []
    right = []
    for i in range(1, len(s)):
        if s[i] < s[0]:
            left.append(s[i])
        if s[i] > s[0]:
            right.append(s[i])
    root.left = buildtree(left)
    root.right = buildtree(right)
    return root

def post(node: Treenode):
    if not node:
        return []
    out = []
    out.extend(post(node.left))
    out.extend(post(node.right))
    out.append(str(node.value))
    return out

n = int(input())
str0 = list(map(int, input().split()))
print(' '.join(post(buildtree(str0))))
```

基本信息

#: 44447209

题目: 22275

提交人: Kelvin

内存: 4076kB

时间: 30ms

语言: Python3

提交时间: 2024-03-29 17:38:24

05455: 二叉搜索树的层次遍历

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

思路:

同第一题类似，先根据二叉搜索树的性质建树，然后借助栈实现分层遍历

代码

```

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

def buildtree(s):
    if len(s) == 0:
        return None
    root = Treenode(s[0])
    left = []
    right = []
    for i in range(1, len(s)):
        if s[i] < s[0]:
            left.append(s[i])
        if s[i] > s[0]:
            right.append(s[i])
    root.left = buildtree(left)
    root.right = buildtree(right)
    return root

def output(root):
    stack = [root]
    out = []
    while stack:
        out.append(str(stack[0].value))
        if stack[0].left:
            stack.append(stack[0].left)
        if stack[0].right:
            stack.append(stack[0].right)
        stack.pop(0)
    return out

str0 = list(map(int, input().split()))
print(' '.join(output(buildtree(str0))))

```

代码运行截图

状态: Accepted

源代码

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

def buildtree(s):
    if len(s) == 0:
        return None
    root = Treenode(s[0])
    left = []
    right = []
    for i in range(1, len(s)):
        if s[i] < s[0]:
            left.append(s[i])
        if s[i] > s[0]:
            right.append(s[i])
    root.left = buildtree(left)
    root.right = buildtree(right)
    return root

def output(root):
    stack = [root]
    out = []
    while stack:
        out.append(str(stack[0].value))
        if stack[0].left:
            stack.append(stack[0].left)
        if stack[0].right:
            stack.append(stack[0].right)
        stack.pop(0)
    return out

str0 = list(map(int, input().split()))
print(' '.join(output(buildtree(str0))))
```

基本信息

#: 44447215
题目: 05455
提交人: Kelvin
内存: 3680kB
时间: 25ms
语言: Python3
提交时间: 2024-03-29 17:38:52

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：

参照课件中的代码来写

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]:
                tmp = self.heaplist[i]
                self.heaplist[i] = self.heaplist[i//2]
                self.heaplist[i//2] = tmp
            i = i // 2

    def insert(self, k):
        self.heaplist.append(k)
        self.currentSize += 1
        self.percUp(self.currentSize)

    def minChild(self, i):
        if 2 * i + 1 > self.currentSize:
            return 2 * i
        else:
            if self.heaplist[2 * i] < self.heaplist[2 * i + 1]:
                return 2 * i
            else:
                return 2 * i + 1

    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

    def delMin(self):
        delmin = self.heaplist[1]
        self.heaplist[1] = self.heaplist[self.currentSize]
        self.currentSize -= 1
        self.heaplist.pop()
        self.percDown(1)
```

```

        return delmin

def buildHeap(self, alist):
    i = len(alist) // 2
    self.currentSize += len(alist)
    self.heaplist.extend(alist)
    while i > 0:
        self.percDown(i)
        i -= 1

bh = BinHeap()
n = int(input())
for i in range(n):
    ope = list(map(int, input().split()))
    if ope[0] == 1:
        bh.insert(ope[1])
    if ope[0] == 2:
        print(bh.delMin())

```

代码运行截图

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]:
                tmp = self.heaplist[i]
                self.heaplist[i] = self.heaplist[i//2]
                self.heaplist[i//2] = tmp
            i = i // 2

    def insert(self, k):
        self.heaplist.append(k)
        self.currentSize += 1
        self.percUp(self.currentSize)

    def minChild(self, i):
        if 2 * i + 1 > self.currentSize:
            return 2 * i
        else:
            if self.heaplist[2 * i] < self.heaplist[2 * i + 1]:
                return 2 * i
            else:
                return 2 * i + 1

    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

    def delMin(self):
        delmin = self.heaplist[1]
        self.heaplist[1] = self.heaplist[self.currentSize]
        self.currentSize -= 1
        self.heaplist.pop()
        self.percDown(1)
        return delmin

    def buildHeap(self, alist):
        i = len(alist) // 2
        self.currentSize += len(alist)
        self.heaplist.extend(alist)
        while i > 0:
            self.percDown(i)
            i -= 1

bh = BinHeap()
n = int(input())
for i in range(n):
    ope = list(map(int, input().split()))
    if ope[0] == 1:
        bh.insert(ope[1])
    if ope[0] == 2:
        print(bh.delMin())
```

基本信息

#: 44449346
题目: 04078
提交人: Kelvin
内存: 4736kB
时间: 682ms
语言: Python3
提交时间: 2024-03-29 19:44:12

22161: 哈夫曼编码树

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

思路：

利用上一题二叉堆的结构来建树，有些复杂，也是参考着课件给的代码来写的

代码

```

#
import heapq

class Node:
    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 buildtree(char_weight_in):
    heap = []
    for char, weight in char_weight_in.items():
        heapq.heappush(heap, Node(weight, char))
    while len(heap) > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)
        merged = Node(left.weight+right.weight, min(left.char, right.char))
        merged.left = left
        merged.right = right
        heapq.heappush(heap, merged)
    return heap[0]

def encode_tree(root):
    codes = {}

    def traverse(node, code):
        if node.left is None and node.right is None:
            codes[node.char] = code
        else:
            traverse(node.left, code+'0')
            traverse(node.right, code+'1')
    traverse(root, '')
    return codes

```

```

def encode(codes, str_input):
    out = ''
    for char in str_input:
        out += codes[char]
    return out

def decode(root, str_input):
    out = ''
    node = root
    for char in str_input:
        if char == '0':
            node = node.left
        if char == '1':
            node = node.right
        if node.left is None and node.right is None:
            out += node.char
            node = root
    return out

n = int(input())
char_weight = {}
for _ in range(n):
    char_in, weight_in = input().split()
    char_weight[char_in] = int(weight_in)
root_tree = buildtree(char_weight)
codes_tree = encode_tree(root_tree)
while True:
    try:
        str_in = input()
        if str_in[0] in ('0', '1'):
            print(decode(root_tree, str_in))
        else:
            print(encode(codes_tree, str_in))
    except EOFError:
        break

```

代码运行截图

状态: **Accepted**

源代码

```
import heapq

class Node:
    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 buildtree(char_weight_in):
    heap = []
    for char, weight in char_weight_in.items():
        heapq.heappush(heap, Node(weight, char))
    while len(heap) > 1:
        left = heapq.heappop(heap)
        right = heapq.heappop(heap)
        merged = Node(left.weight+right.weight, min(left.char, right.char))
        merged.left = left
        merged.right = right
        heapq.heappush(heap, merged)
    return heap[0]

def encode_tree(root):
    codes = {}

    def traverse(node, code):
        if node.left is None and node.right is None:
            codes[node.char] = code
        else:
            traverse(node.left, code+'0')
            traverse(node.right, code+'1')
    traverse(root, '')
    return codes

def encode(codes, str_input):
    out = ''
    for char in str_input:
        out += codes[char]
    return out

def decode(root, str_input):
    out = ''
    node = root
    for char in str_input:
        if char == '0':
            node = node.left
        if char == '1':
            node = node.right
        if node.left is None and node.right is None:
            out += node.char
            node = root
    return out
```

基本信息

#: 44459824

题目: 22161

提交人: Kelvin

内存: 3664kB

时间: 24ms

语言: Python3

提交时间: 2024-03-30 12:22:26

```
n = int(input())
char_weight = {}
for _ in range(n):
    char_in, weight_in = input().split()
    char_weight[char_in] = int(weight_in)
root_tree = buildtree(char_weight)
codes_tree = encode_tree(root_tree)
while True:
    try:
        str_in = input()
        if str_in[0] in ('0', '1'):
            print(decode(root_tree, str_in))
        else:
            print(encode(codes_tree, str_in))
    except EOFError:
        break
```

晴问9.5: 平衡二叉树的建立

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

思路:

手动搭建AVL树结构, 非常复杂, 还是参考着课件来写的

代码

```

#
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

平衡二叉树的建立

通过数 330 提交数 824

题目描述

将 n 个互不相同的正整数。

输入描述

第一行一个整数 n ($1 \leq n \leq 1000$)

第二行 n 个整数 a_i ($1 \leq a_i \leq 10000$)

输出描述

输出 n 个整数，表示先序遍历序列。

样例1

输入 复制

```
5
5 2 3 6 8
```

输出 复制

```
3 2 6 5 8
```

解释

插入的过程如下图所示。

完美通过

100% 数据通过测试

运行时长: 0 ms

语言: Python

```

1 class Node:
2     def __init__(self, value):
3         self.value = value
4         self.left = None
5         self.right = None
6         self.height = 1
7
8
9 class AVL:
10     def __init__(self):
11         self.root = None
12
13     def insert(self, value):
14         if not self.root:
15             self.root = Node(value)
16         else:
17             self.root = self._insert(value, self.root)
18
19     def _insert(self, value, node):
20         if not node:
21             return Node(value)
22         elif value < node.value:
23             node.left = self._insert(value, node.left)
24         else:
25             node.right = self._insert(value, node.right)
26
27     def _rotate_right(self, node):
28         if value < node.left.value: # 树形是 LL
29             return self._rotate_right(node)
30         else: # 树形是 LR
31             node.left = self._rotate_left(node.left)
32             return self._rotate_right(node)
33
34     def _rotate_left(self, node):
35         if value > node.right.value: # 树形是 RR
36             return self._rotate_left(node)
37         else: # 树形是 RL
38             node.right = self._rotate_right(node.right)
39             return self._rotate_left(node)
40
41     def _balance(self, node):
42         balance = self._get_balance(node)
43         if balance > 1:
44             if value > node.right.value: # 树形是 RR
45                 return self._rotate_left(node)
46             else: # 树形是 RL
47                 node.right = self._rotate_right(node.right)
48                 return self._rotate_left(node)
49         elif balance < -1:
50             if value < node.left.value: # 树形是 LL
51                 return self._rotate_right(node)
52             else: # 树形是 LR
53                 node.left = self._rotate_left(node.left)
54                 return self._rotate_right(node)
55         return node
56
57     def _get_balance(self, node):
58         if not node:
59             return 0
60         left_height = self._get_height(node.left)
61         right_height = self._get_height(node.right)
62         return left_height - right_height
63
64     def _get_height(self, node):
65         if not node:
66             return 0
67         return max(self._get_height(node.left), self._get_height(node.right)) + 1
68
69     def preorder(self):
70         self._preorder(self.root)
71
72     def _preorder(self, node):
73         if not node:
74             return
75         print(str(node.value) + " ", end='')
76         self._preorder(node.left)
77         self._preorder(node.right)
78
79     def __str__(self):
80         return self.preorder()

```

结果 历史提交

结果	时长(ms)	语言	查看
完美通过	0	Python	查看
编译错误	0	C++	查看

收起面板 运行 提交

02524: 宗教信仰

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

思路：

利用并查集中father函数的思想对宗教信仰进行分类

代码


```

#
def init_set(n):
    return list(range(n))

def get_father(x, father):
    if father[x] != x:
        father[x] = get_father(father[x], father)
    return father[x]

def join(x, y, father):
    fx = get_father(x, father)
    fy = get_father(y, father)
    if fx == fy:
        return
    father[fx] = fy

case_num = 0
while True:
    n, m = map(int, input().split())
    if n == 0 and m == 0:
        break
    count = 0
    father = init_set(n)
    for _ in range(m):
        s1, s2 = map(int, input().split())
        join(s1 - 1, s2 - 1, father)
    for i in range(n):
        if father[i] == i:
            count += 1
    case_num += 1
    print(f"Case {case_num}: {count}")

```

代码运行截图

状态: Accepted

源代码

```
def init_set(n):
    return list(range(n))

def get_father(x, father):
    if father[x] != x:
        father[x] = get_father(father[x], father)
    return father[x]

def join(x, y, father):
    fx = get_father(x, father)
    fy = get_father(y, father)
    if fx == fy:
        return
    father[fx] = fy

case_num = 0
while True:
    n, m = map(int, input().split())
    if n == 0 and m == 0:
        break
    count = 0
    father = init_set(n)
    for _ in range(m):
        s1, s2 = map(int, input().split())
        join(s1 - 1, s2 - 1, father)
    for i in range(n):
        if father[i] == i:
            count += 1
    case_num += 1
    print(f"Case {case_num}: {count}")
```

基本信息

#: 44453068
题目: 02524
提交人: Kelvin
内存: 5840kB
时间: 1246ms
语言: Python3
提交时间: 2024-03-29 22:56:16

2. 学习总结和收获

本次作业前两题还好，后面四道题对我而言比较困难，分别考察了二叉堆、哈夫曼编码算法、AVL树和并查集，做的时候参考着课件给的代码一行一行理解再手动敲一遍，深刻感受到了算法的精妙。考试的时候得把这部分的代码打印下来带进去。